INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Multi-Agent Systems Development to Control the LCAR Manufacturing Cell

José Pedro Sampaio da Costa

Dissertation presented to *Escola Superior de Tecnologia e Gestão* of the *Instituto Politécnico de Bragança* to obtain an MSc in Electrical and Computer Engineering.

Work oriented by:

PhD José Barbosa

PhD Paulo Leitão

This document doesn't include the suggestions made by the board.

Bragança, Portugal

2024

Multi-Agent Systems Development to Control the LCAR Manufacturing Cell

José Pedro Sampaio da Costa

Dissertation presented to *Escola Superior de Tecnologia e Gestão* of the *Instituto Politécnico de Bragança* to obtain an MSc in Electrical and Computer Engineering.

Work oriented by:

PhD José Barbosa

PhD Paulo Leitão

This document doesn't include the suggestions made by the board.

Bragança, Portugal

2024

Preface

The dissertation was prepared at *Escola Superior de Tecnologia e Gestão* (ESTiG) in fulfillment of the requirements for acquiring an MSc in Electrical and Computer Engineering.

The project was carried out in 2024 for 42 ECTS points, from February 12th to September 10th, with José Barbosa and Paulo Leitão as supervisors.

The dissertation concerns the development and evaluation of a Multi-Agent System (MAS) using the Java Agent Development Framework (JADE).

The MAS targets the Small-Scale Production System scenario located in the *Laboratório de Controlo, Automação e Robótica* (LCAR).

The dissertation aims to develop a monitoring and control system for Small-Scale Production, with each device considered an agent.

Familiarity with Object-Oriented Language (OOL), such as Java Programming Language, the obstacle has been overcome to writing the dissertation. Knowledge about Artificial Intelligence (AI) and Distributed Artificial Intelligence (DAI) is also helpful, although the necessary concepts are introduced in the dissertation.

Acknowledgment

Here I would like to thank everyone who contributed to the development of this dissertation.

Firstly, I would like to express my most profound gratitude to Adolfo Rodrigues da Costa, my father, and Elsa Cristina Leite, my mother. Without them, without their unfailing support in difficult times, I would never have been studying for a Master's, and this document would be blank. I hope they are very proud of me and my work.

I want to thank my girlfriend Ana Filipa Martinho for all the support, advice, help and especially for making this work lighter with her contagious joy.

Secondly, I want to thank my supervisors, of the *Instituto Politécnico de Bragança* (IPB), PhD José Barbosa, and PhD Paulo Leitão for their support, document reviews, criticisms, discussions, and suggestions, contributing directly to the achieved results. I'm very lucky to have all these people with me who gave me a lot of support to complete this project.

Thirdly, I would like to thank the IPB community for making the LCAR available for this dissertation.

Finally, I would also like to acknowledge people not mentioned here who, whether directly or indirectly, contributed to the writing of this document.

Abstract

This dissertation aims to design and implement a monitoring and control system for a small-scale production system, guided by the imperatives of Cyber-Physical Systems (CPS) and Industry 4.0.

One of the main objectives of Industry 4.0 is the connectivity of devices and systems using Internet of Things (IoT) technologies, and CPS that acts as the backbone of an infrastructure based on distributed and decentralized structures. CPS requires the use of Distributed Artificial Intelligence (DAI) techniques, such as agents and Multi-Agent Systems (MAS). The most fundamental concept of agents is to view them as entities that emulate human-like attributes, such as decision-making, intention, and perception. Agents have actions, behaviors, beliefs, intentions, and desires just like humans. CPS requires the use of MAS, allowing the incorporation of intelligence into the CPS through autonomous, proactive, and cooperative actions.

The case study focuses on a small-scale production system, consisting of an IRB 1400 ABB robot, two punching machines, and two indexed lines supplied by Fischertechnik^{TM} which runs in a Modicon M340 PLC.

The dissertation aims to develop an MAS in JADE to control the production system, with each device considered as an agent. The agents communicate with each other and report their current status, actions, and behaviors through messages formatted according to the FIPA-ACL specification. A classic approach will be adopted: requirements analysis, MAS modeling, implementation, and testing.

Keywords: Cyber-Physical Systems · FIPA · Industry 4.0 · JADE · MODBUS TCP/IP · MQTT Broker · OPC-UA

Resumo

Esta dissertação visa projetar e implementar um sistema de monitorização e controlo para um sistema de produção de pequena escala, orientado pelos imperativos dos Sistemas Ciberfísicos (SC) e da Indústria 4.0.

Um dos principais objetivos da Indústria 4.0 é a conetividade de dispositivos e sistemas utilizando tecnologias da Internet das Coisas (IoT) e SC que atuam como a espinha dorsal de uma infraestrutura baseada em estruturas distribuídas e descentralizadas. Os SC requerem o uso de técnicas de Inteligência Artificial Distribuída (IAD), tais como agentes e Sistemas Multiagentes (SMA). O conceito mais fundamental de agentes é vê-los como entidades que imitam atributos semelhantes aos humanos, como a tomada de decisões, a intenção e a perceção. Os agentes têm ações, comportamentos, crenças, intenções e desejos, tal como os humanos. Os SC requerem o uso de SMA, permitindo a incorporação de inteligência nos SC via ações autónomas, proativas e cooperativas.

O caso de estudo incide sobre um sistema de produção de pequena escala, constituído por um robô IRB 1400 ABB, duas máquinas de punção e duas linhas indexadas fornecidas pela Fischertechnik^{TM} que funciona num PLC Modicon M340.

A dissertação visa desenvolver um SMA em JADE para controlar o sistema de produção, com cada dispositivo considerado um agente. Os agentes comunicam entre si e reportam o seu estado atual, ações e comportamentos por mensagens formatadas conforme a especificação FIPA-ACL. Irá ser adotada uma abordagem clássica de: análise de requisitos, modelação do SMA, implementação e teste.

Palavras-chave: FIPA · Indústria 4.0 · JADE · MODBUS TCP/IP · MQTT Broker · Sistemas Ciber-Físicos · OPC-UA

Contents

List of Tables

List of Figures

Listings

Acronyms

ABM Agent-Based Modelling.

ACC Agent Communication Channel.

ACL Agent Communication Language.

ADU Application Data Unit.

AI Artificial Intelligence.

AID Agent Identifier.

AMS Agent Management System.

AOP Agent-Oriented Programming.

AP Agent Platform.

API Application Programming Interface.

BDI Belief-Desire-Intention.

BRF Belief Revision Function.

CLI Java Command-Line Interface.

CORBA Common Object Request Broker Architecture.

CPS Cyber–Physical Systems.

DAI Distributed Artificial Intelligence.

DBMS Data Base Management Systems.

DCA Distributed Computing Architecture.

DF Directory Facilitator.

DMAS Distributed Multi-Agent System.

DPS Distributed Problem Solving.

ESTiG *Escola Superior de Tecnologia e Gestão.*

FIPA Foundation for Intelligent Physical Agents.

FIPA-ACL FIPA Agent Communication Language Specifications.

GUI Graphical User Interface.

HTTP Hyper Text Transfer Protocol.

IAC Inter-Agent Communication.

IAD *Inteligência Artificial Distribuída.*

IIOP Internet Inter-ORB Protocol.

IMTP Internal Message Transport Protocol.

IoT Internet of Things.

IP Internet Protocol.

IPB *Instituto Politécnico de Bragança.*

IPMT Internal Platform Message Transport.

JADE Java Agent DEvelopment Framework.

JAS Java Agent Services.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

LCAR *Laboratório de Controlo, Automação e Robótica.*

MAA Multi-Agent Architecture.

MAE Multi-Agent Environment.

MAL Multi-Agent Learning.

MAP Multi-Agent Platform.

MAS Multi-Agent Systems.

MBAP Header MODBUS Application Header.

MQTT Message Queuing Telemetry Transport.

MTP Message Transport Protocol.

MTS Message Transport Service.

OOL Object-Oriented Language.

OPC-UA OPC Unified Architecture.

PCB Printed Circuit Board.

PDU Protocol Data Unit.

PI Parallel Intelligence.

PLC Programmable Logic Controllers.

QoS Quality of Service.

RMA Remote Monitoring Agent.

RMI Java Remote Method Invocation.

RTU Remote Terminal Unit.

SC	*Sistemas Ciberfísicos.*
SCADA	Data Acquisition and Monitoring Control System.
SL	Semantic Language.
SMA	*Sistemas Multiagentes.*
SSL	Secure Sockets Layer.
TCP	Transmission Control Protocol.
UML	Unified Modeling Language.
URL	Uniform Resource Locator.

Chapter 1

Introduction

This chapter presents the motivation and specific objectives for the development of a monitoring and control system for small-scale production. The objective is to utilize Java Agent DEvelopment Framework (JADE), to design and develop Multi-Agent Systems (MAS) for monitoring and control of the small-scale production system.

1.1 Motivation

According to [14] Cyber–Physical Systems (CPS) constitute the backbone to implement the Industry 4.0 principles, enabling the next generation of production systems, that are characterized by the integration, in a network and large-scale, of several heterogeneous, connected, and smart processes systems, and devices, combining the cyber and physical counterparts, which allows for improving the production systems performance, responsiveness, and reconfigurability, while efficiently managing the complexity and uncertainty.

According to [7] it can be said that a simple agent has very little intelligence because its actions are closely tied to perception. Examples from nature are insects, whose range of behavior is minimal, but they are robust and difficult to eliminate due to their high numbers. Even the behavior of more complex creatures, frogs, is based on behavior. For example, it turns out that the frog's visual system is tightly coupled with the response to fly-sized objects moving at roughly the speed of a fly. The frog automatically sticks out its tongue in an attempt to catch a flying object.

According to what is described in [1, 10] it can be said that MAS consists of several interacting agents, observing and acting upon an environment. By coordinating intelligent agents, it's possible to find solutions to problems that individual agents cannot. Intelligent agents are often characterized by the properties of autonomy, proactiveness, reactivity, and social ability. As a result, the individual agents are capable of goal-based reasoning while coordinating their efforts toward a common solution and reacting to sudden changes.

Implementing efficient coordination is considered the most challenging task concerning MAS, hence many tools to facilitate the development of such systems. These include the Belief-Desire-Intention (BDI) and the JADE which builds upon in Java Programming Language. By exploiting these tools and the concepts that follow, an attempt will be made to develop a highly sophisticated MAS capable of solving a high level of coordination.

1.2 Specific Objectives

The dissertation aims to develop a monitoring and control system for the small-scale production system located at the *Laboratório de Controlo, Automação e Robótica* (LCAR) of the *Escola Superior de Tecnologia e Gestão* (ESTiG) of the *Instituto Politécnico de Bragança* (IPB).

The beginning of the study consists of studying the concept of agent and, in particular, MAS with industrial applications.

The development is carried out using agent systems technology, specifically the JADE framework.

In simplified terms, distributed software components are designed and developed to represent the existing physical components, particularly the Fischertechnik^{TM} production machines and the IRB 1400 ABB Robot for handling.

The interactions between the various agents are also designed to support the realization of production tasks, simulating a control system. Each agent will also send information about the state and other variables of the system, realizing the monitoring functionality.

1.3 Dissertation Structure

This dissertation is organized into 6 chapters, beginning with the present chapter where the proposal of the work, contextualization, and objectives are presented. The dissertation is structured as follows:

- Chapter 2 introduces the term agent, environment, MAS and industrial applications, and provides relevant background knowledge;
- Chapter 3 introduces the Agent-Oriented Programming (AOP), JADE and communication protocols among agents such as Foundation for Intelligent Physical Agents (FIPA), Agent Communication Language (ACL), and communication protocol sub as MODBUS, MQTT, and OPC-UA from a theoretical viewpoint. It describes how to exchange messages between agents using Internal Message Transport Protocol (IMTP);
- Chapter 4 describes the Small-Scale Production System scenario and the agents used, such as Client Agent, Product Agent, and Resource Agent;
- Chapter 5 describes the results obtained and discusses them from the Small-Scale Production System scenario, described in chapter 4;
- Finally, chapter 6 concludes the dissertation.

To contextualize the theoretical issues addressed in this dissertation, the following appendices are attached:

- Appendix A which includes the Unified Modeling Language (UML) diagram for the Small-Scale Production System scenario;
- Appendix B which includes all the Java Programming Language for the Small-Scale Production System scenario.

Chapter 2

Theoretical Background

This chapter aims at the intersection of CPS and Industry 4.0. Definitions of key terms like Distributed Artificial Intelligence (DAI), and MAS are provided, highlighting their significance in advancing automation. The chapter further examines agent classifications, architectures, environment, communication, coordination, and applications in MAS.

2.1 Cyber-Physical Systems and Industry 4.0

The 4th Industrial Revolution, also known as Industry 4.0 is driven by innovative technologies that cause changes in production systems, as shown in Figure 2.1.

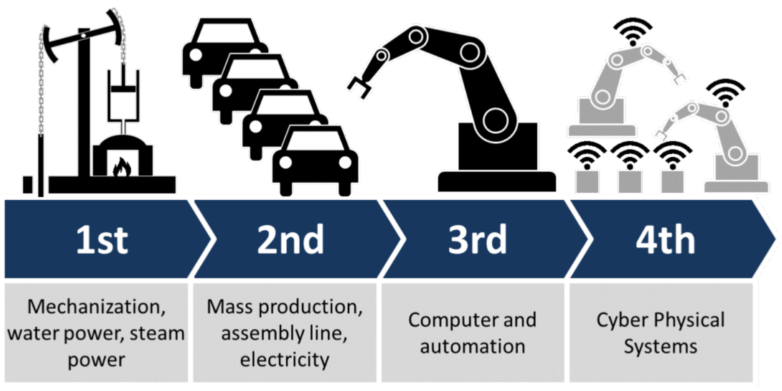

1st
2nd
3rd
4th
Mechanization, water power, steam power
Mass production, assembly line, electricity
Computer and automation
Cyber Physical Systems

Figure 2.1: Evolution of Systems in Industrial Revolutions [5].

This revolution promotes the connectivity of devices and systems through the use of Internet of Things (IoT) technologies, complemented with the use of other technologies, such as Big Data, Cloud Computing, and Artificial Intelligence (AI).

According to [14, 15] CPS is a backbone approach to developing smart systems under decentralized structures, based on a network of cyber and physical entities that are integrated to realize an objective. CPS can be seen as a way to transform traditional factories into intelligent factories in Industry 4.0. CPS is more involved with collaborative activities of sensors and actuators to achieve certain objectives and for that, IoT technologies are used to reach collaborative work on distributed systems.

An overview of the CPS can be seen in Figure 2.2, which shows the requirements, applications, and what these systems are. Pays great attention to applications in energy and different forms of manufacturing.

2.2 Definitions

Before going into detail about MAS, it's important to understand the meaning of DAI, it's also important to define what an agent is and why it's considered intelligent.

2.2.1 Distributed Artificial Intelligence

According to [10] DAI is a sub-field of AI that has gained considerable importance due to its ability to solve problems in the real world.

The primary objective in the field of DAI has included three different areas which are Parallel AI, Distributed Problem Solving (DPS), and MAS [16, 17].

Parallel Intelligence (PI) refers to approaches that make classical AI techniques easier to apply to Distributed Computing Architecture (DCA), such as cluster-based or multiprocessor computing. The Parallel AI aims to increase the operation speed and work on parallel threads to arrive at a global solution for a particular problem.

The DPS is similar to Parallel AI and considers how a problem can be solved by sharing the resources between many cooperating modules, known as a computing entity. In DPS, communication between computing entities and, the quantity of information shared are predetermined and embedded in the design of the computing

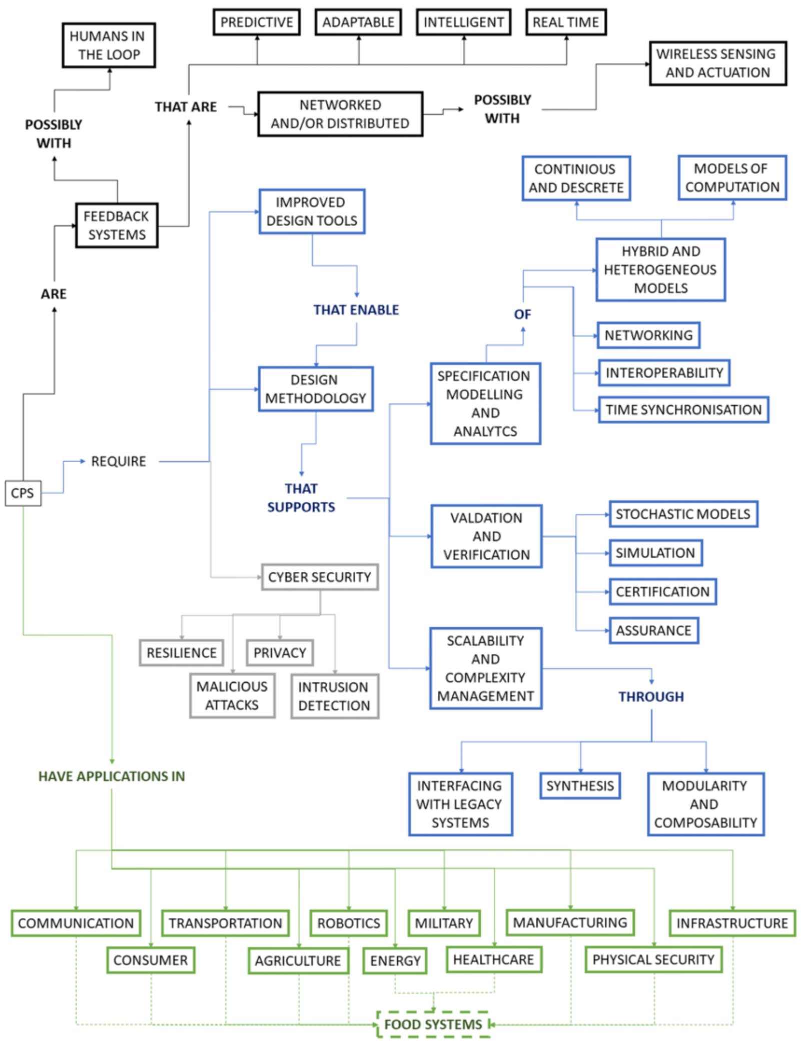


Figure 2.2: CPS Concept Map [6].

entity. DPS is rigid due to the embedded strategies and offers little or no flexibility.

In contrast to DPS, MAS deals with the behavior of the computing entities available to solve a given problem. Every computing entity in MAS is called an agent. A network of individual agents that collaborate to solve problems by exchanging information and communicating with one another is referred to as MAS [18, 19].

Since the control system is distributed, each agent has limited capabilities and information, data is decentralized, and the computations are asynchronous, the only way to guarantee the desired performance of DAI systems is to allow appropriate interactions between agents. Figure 2.3 illustrates a Terminological Background about each term discussed previously in this section.

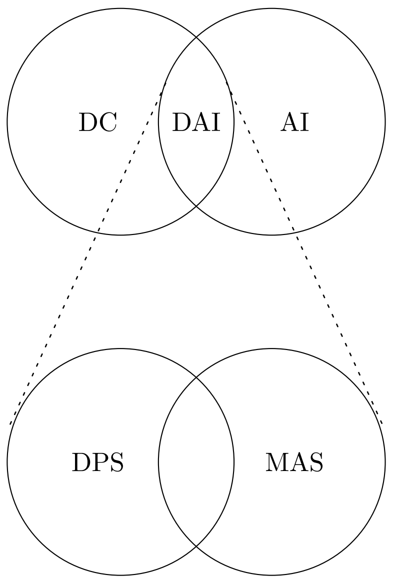


Figure 2.3: DAI Illustration (Adapted from [7]).

2.2.2 Agent

One of the most important paradigms that could improve methods for implementing software systems and provide a solution for the problem of integrating legacy software is agents.

An agent cannot be defined universally because various issues need distinct domain-specific aspects to be accurately depicted. The most abstract representation of an agent, however, is illustrated in Figure 2.4, where an agent acts via actions upon the environment and uses percepts to perceive its state [8].

According to [20] with its actuators, the agent may perform on its environment, and its sensors pick up changes in that environment. The agent's internal workings determine how the agent reacts to different impressions and what actions it should take next.

The term "Agent" has been used in various technologies, including AI, Data Base Management Systems (DBMS), computer networks, and operating systems [1]. While there isn't a single definition for an agent, [21] describes an agent as essentially a unique autonomous software component that acts as a compatible interface to any system and behaves like a human agent, pursuing its goals on behalf of some clients. An agent system is often composed of numerous agents, however, it can also be based on a single agent operating in an environment and, if necessary, interacting

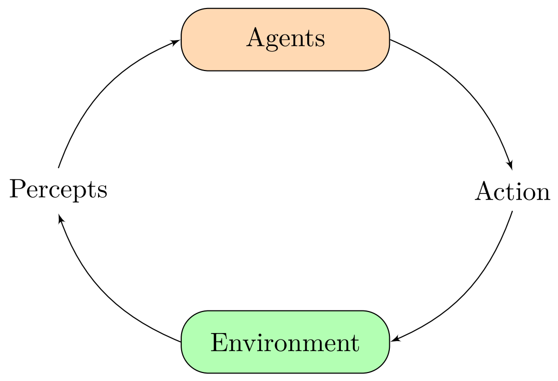


Figure 2.4: A Model of an Agent and its Environment [8].

with its users. These systems can introduce the idea of agents sharing goals and communicating with each either directly or indirectly.

Figure 2.5 differentiates between a weak agent and a strong agent, it can be said that a weak agent is a hardware or software that's situated in a given environment, and which is capable of acting autonomously in that environment to fulfill its delegated objectives. In the definition of a strong agent, terms characterizing "mental" and "emotional" states are added to the properties mentioned [7].

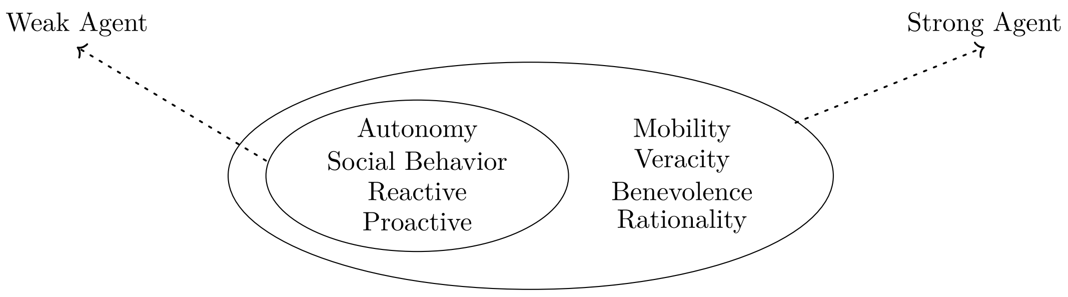


Figure 2.5: Weak Agent vs. Strong Agent [7].

According to [10] expert systems and distributed controllers can be distinguished from agents. Several important features that set an agent apart from basic controllers will be covered in the following discussion.

- **Autonomy**, can be defined as the ability of an agent to choose its actions independently without external intervention by other agents in the network.

- **Inferential capability**, the ability of an agent to work on abstract goal specifications such as deducing an observation by generalizing the information.

- **Responsiveness**, the ability to perceive the condition of the environment and respond to it in a timely fashion to take account of any changes in the environment. This property is of critical importance in real-time applications.
- **Pro-activeness**, the agent must exhibit a good response to opportunistic behavior. This is to enhance goal-directed actions rather than just being responsive to a specific change in environment. It must have the ability to adapt to any changes in the dynamic environment.
- **Social behavior**, even though the agent's decision must be free from external intervention, it must still be able to interact with external sources when the need arises to achieve a specific goal. It must also be able to share this knowledge and help other agents solve a specific problem.

The most widely accepted definition of an agent is an adaptable, independent being with the ability to sense its surroundings via the sensors attached to it. Through actuators, these affect their surroundings.

To be able to respond to and act in a dynamic, non-deterministic environment, an agent needs to have at least two of the three qualities of autonomy: cooperation; learning; and the capacity to operate independently in the environment without human aid. Through the agents' interactions with the surroundings, this learning component is incorporated. An agent that wants to interact with other agents can utilize a set of social abilities called cooperation. It is believed that learning over time can improve how the agents in the environment behave.

2.2.3 Environment

The environment can be divided into the following categories: partially or fully observable, static or dynamic, episodic or sequential, deterministic or stochastic, and discrete or continuous. Where each of these categories will be explained below.

Figure 2.6 illustrates how an agent interacts with its environment. The agent receives an indication of the environment's state. The environment state is mapped into perceptual inputs, which provide information to the agent about the environment.

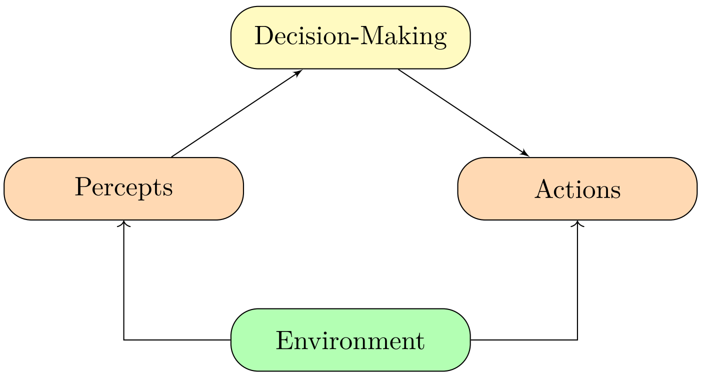


Figure 2.6: Agent-Environment Interaction [9].

Partially Observable or Fully Observable

If an agent can view the complete state at any given time, then an environment is fully observable. Since the agent does not need to maintain an internal state, working with a fully observable environment is more convenient.

A setting may also be unobservable, in which case the environment is unknown to the agent [8].

Static or Dynamic

An environment is considered static when it stays the same unless an agent acts upon it. If the world can change without the influence of an agent, it's considered dynamic.

An agent finds it easier to work with a static environment because it doesn't need to constantly watch it to know which state it's in.

The environment in Multi-Agent Environment (MAE) may appear static to individual agents, but it may be dynamic. Other agents may act upon and modify the environment while an agent is deciding what to do [8].

Episodic or Sequential

In an episodic environment, each state is independent of the one before it. Stated differently, agents can effectively restart themselves in each step and their actions have no lasting impact on the environment.

Conversely, in sequential environmentss, every step is dependent upon the one before it, meaning that the actions of agents have an impact on the environment's future states. To accomplish their objectives, agents may need to plan ahead [8].

Deterministic or Stochastic

Every action in a deterministic environment will only have a single impact on the environment, which may already be known at the time of the action. An activity is considered stochastic in the environment if there are several alternative outcomes. This adds ambiguity to the issue. To model this, actions frequently assign a chance to each conceivable outcome.

Agents must be able to adjust and reevaluate their plans in a stochastic environment based on the actual outcomes of their actions [8].

Discrete or Continuous

The actions, percepts, and states that make up a discrete environment are fixed. Conversely, an infinite number of these can exist in a continuous environment [8].

2.2.4 Multi-Agent Systems

According to [10] MAS is an extension of the agent technology where a group of loosely connected autonomous agents acts in an environment to achieve a common goal. MAS have been adapted in many application domains because of the beneficial advantages offered. Therefore, some benefits available to using MAS technology in large systems according to article [22] are:

- An increase in the speed and efficiency of the operation due to parallel computation and asynchronous operation;
- A degradation of the system when one or more of the agents fail. It's thereby increasing the reliability and robustness of the system;
- Flexibility where agents can be added when necessary;
- Reduced cost, this is because individual agents cost much less than a centralized architecture;

- Reusability where agents have a modular structure, and they can be easily replaced in other systems or be upgraded more easily than a monolithic system.

In a MAS, the action of an agent not only modifies its environment and of its neighbors. This means that each agent has to predict the actions of the other agents in order to decide on the optimal course of action to achieve the common goal. This type of concurrent learning could result in non-stable behavior. The problem is further complicated if the environment is dynamic. Then each agent needs to differentiate between the effects caused due to other agent actions and variations in the environment itself [10].

In a Distributed Multi-Agent System (DMAS), the agents are scattered all over the environment. Each agent has a limited sensing capability because of the limited range and coverage of the sensors connected to it. This limits the view available to each of the agents in the environment. Therefore, decisions based on partial observations made by each of the agents could be suboptimal, and achieving a global solution by this means becomes intractable [10].

In an agent system, it's assumed that an agent knows its entire action space and mapping of the state space to action space could be done by experience.

In MAS, to create a map, the agent must be able to learn from the experience of other agents with similar capabilities or decision-making powers. In the case of cooperating agents with similar goals, this can be done easily by creating communication between the agents. In the case of competing agents, it isn't possible to share the information as each of the agents tries to increase its chance of winning. It's therefore essential to quantify how much of the local information and the capabilities of another agent must be known to create improved modeling of the environment [10].

2.3 Agent Classifications

Agent classification is a supplement to agent definitions, explained in 2.2.2. As there are many definitions there are also many agent classifications, but all are according to their properties. Figure 2.7 shows a general classification of agents.

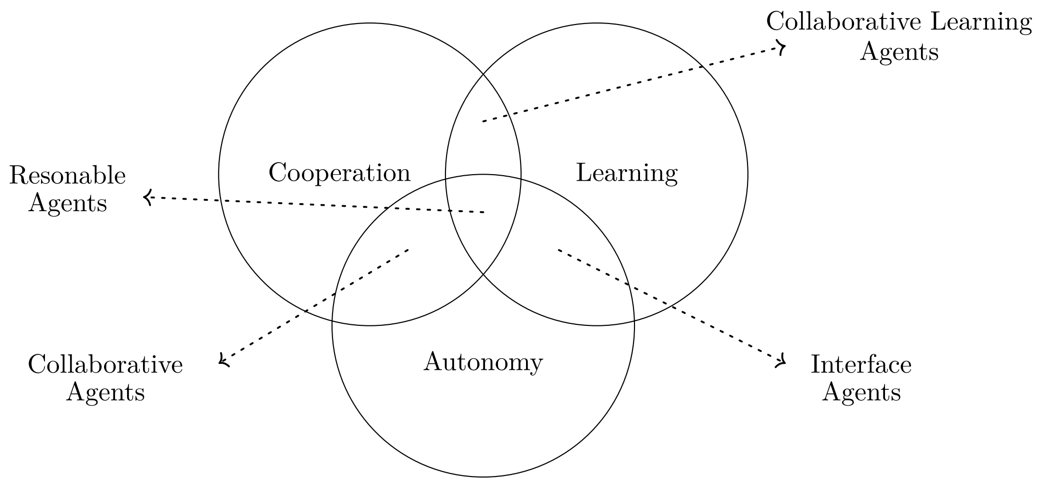


Figure 2.7: Illustration of Agent Classification (Adapted from [7]).

An agent is autonomous when it is self-goal-directed and acts without requiring user initiation and guidance. The agent's behavior is determined by experience, an autonomous agent is a system that is part of the environment it perceives, acts on, and exists in over time, performing its tasks and thus influencing what it will perceive in the future [7].

According to [23] agents can be social, reactive, and deliberative. An agent can be social because the agent interacts with other agents (and possibly humans) via cooperation, coordination, and negotiation. It's able to reason about other agents and how they can help it achieve its own goals.

A simple reactive agent requires almost no memory, it simply implements preprogrammed behaviors. A more complex organism maintains a representation of its environment, thanks to which it remembers the location of unseen objects or the sequence of events. A reactive agent perceives its environment and responds in a timely fashion to changes that occur in the environment [7]. Figure 2.8 exemplifies the operation of a purely reactive agent without any internal state, and Figure 2.9 exemplifies the operation of a reactive agent with an internal state.

An agent can be deliberative/intentional because it doesn't simply act in response to its environment but can exhibit goal-directed behavior by taking initiative [7] as illustrated in Figure 2.10.

An intentional agent can consider its abilities to achieve some goal [7]. A representative theory of deliberative agents is the so-called BDI theory, as explained in

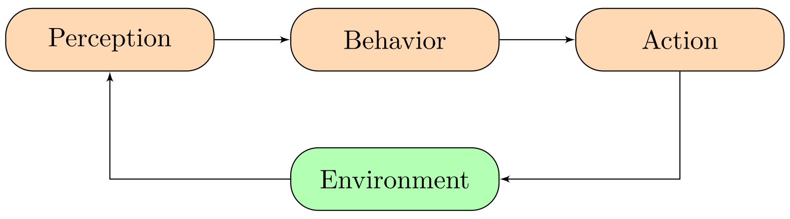


Figure 2.8: Typical Purely Reactive Agent Blocks [7].

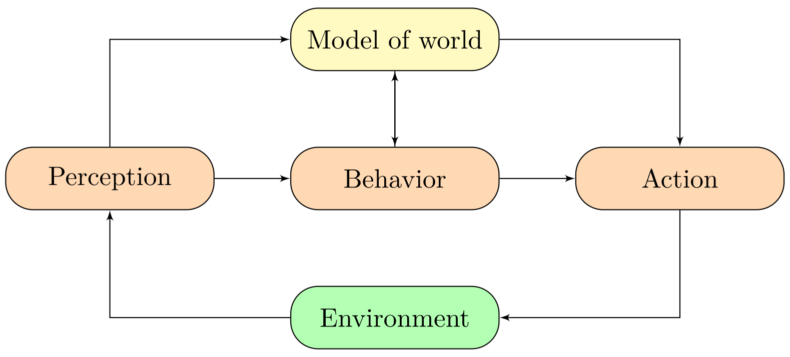


Figure 2.9: Typical Reactive Agent Blocks [7].

3.1.1. The progress from perception to planning is a closed cycle and each activity represents so-called mental categories that are important in terms of intention and higher rationality of behavior.

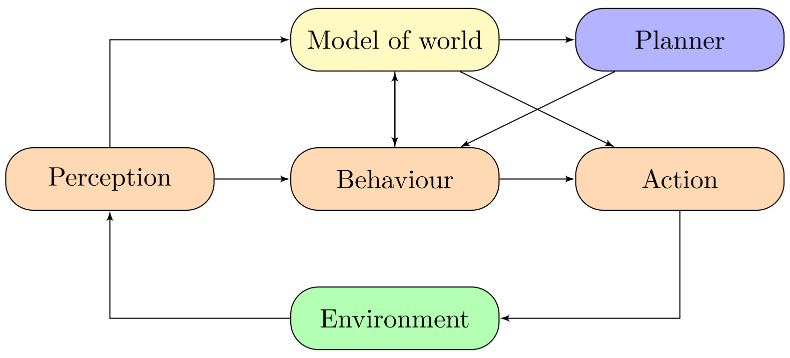


Figure 2.10: Typical Deliberative Agent Blocks [7].

2.4 Classification of Multi-Agent System

MAS classification is a supplement for MAS definitions, explained in 2.2.4, the classification of MAS can be done based on different attributes such as Architecture [24], Learning [25, 26], Communication [24], and Coordination [27]. Figure 2.11 represents a classification of MAS based on various important attributes that define

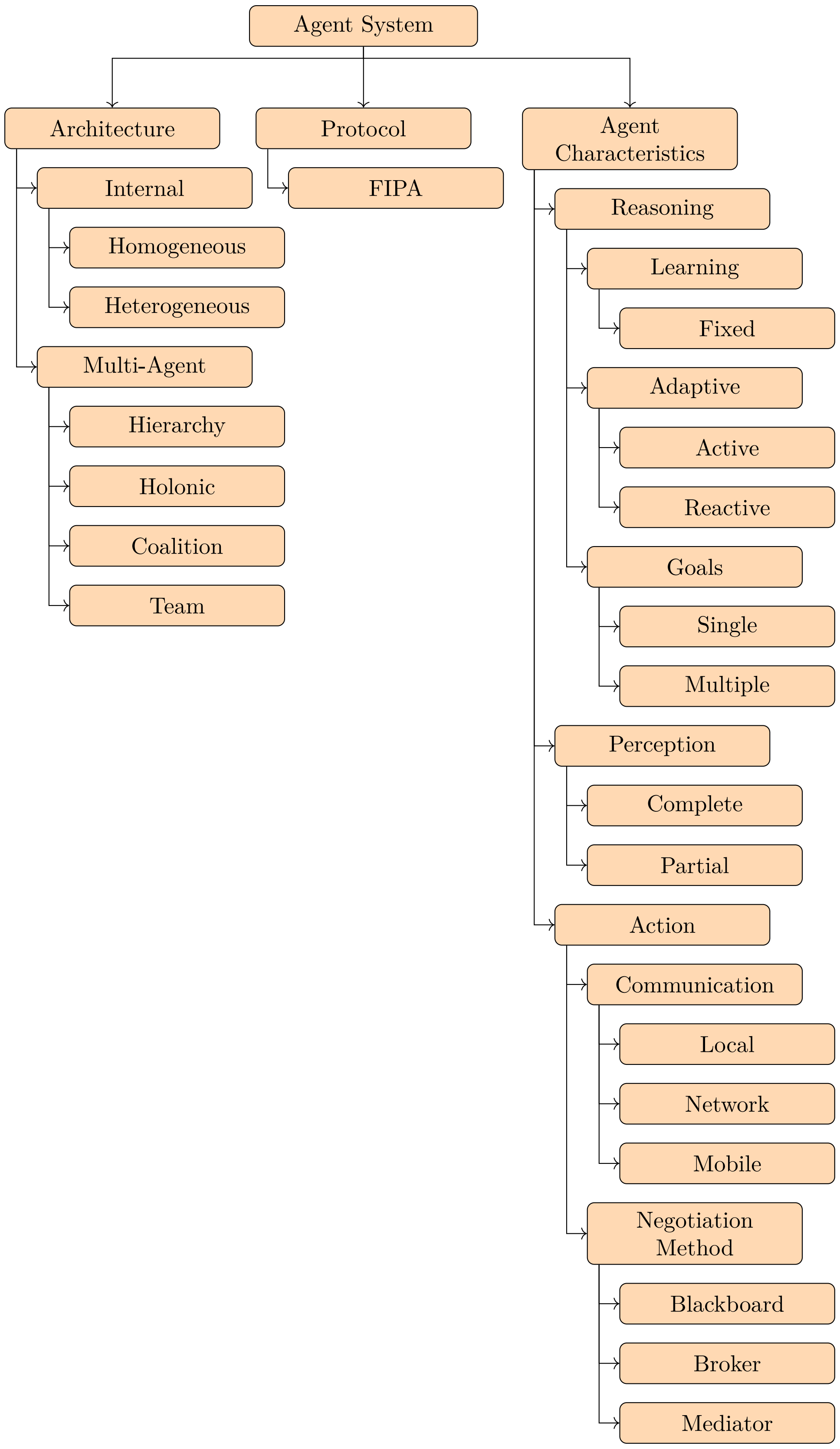


Figure 2.11: Classification of a MAS (Adapted from [10]).

the structure, behavior, and interactions of agents within the system such as agent system, this is a central concept, architecture, it is subdivided into internal architecture and MAS architecture. It also deals with the protocol attribute and agent characteristics that refer to the properties and behaviors of agents.

According to [23], MAS coordination means allocating scarce resources and providing intermediate results. MAS cooperation means that the aim is to improve the quality of work through acceleration of the solution, extensions of the class of solved tasks, increasing the number of solved tasks, and reducing the number of collisions between competing agents. In MAS communication is necessary condition for cooperation and coordination.

2.4.1 Architecture

According to [10] the Internal Architecture of the particular individual agents forming the MAS, may be classified into two types, which are Homogeneous Structure and Heterogeneous Structure, which will be discussed below.

- **Homogeneous Structure:** every agent that makes up the MAS in a homogeneous structure has the same internal architecture. The physical location of the agents and the area of the environment where the action is performed differ from one another. There may be an overlap in the sensor inputs received by each agent, and each agent receives input from various areas of the environment [23, 28, 29].

- **Heterogeneous Structure:** the agents have different capabilities, structures, and functionalities. The actions selected by the agent may differ from those of the agent placed in a different part, but it will still have the same functionality based on the dynamics of the environment and the location of the specific agent. Modeling applications can become much more realistic with the aid of heterogeneous architecture [23, 30].

2.4.2 Overall Agent Organization

The overall organization of agents can be classified as Hierarchical, Holonic, Coalitions, and Teams. These will be explained in detail below.

Hierarchical

One of the oldest organizational designs in MAS is the Hierarchical Organization. The agents are grouped in a conventional tree-like manner within the hierarchical agent architecture [10].

The degree of autonomy varies between agents at various levels in the tree structure. Agents with a higher hierarchy usually receive data from the lower levels of the hierarchy. Supervisory or control signals move from a higher hierarchy to a lower one.

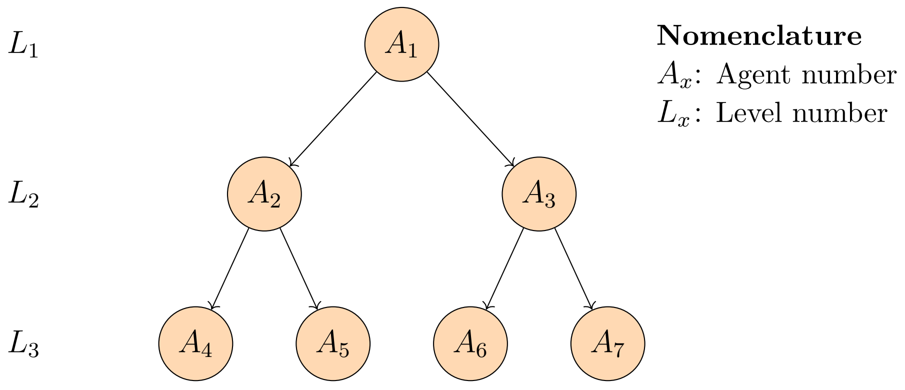


Figure 2.12: Example of Three Hierarchical Agent Architecture (Adapted from [10]).

Figure 2.12 shows a typical three Hierarchical Multi-Agent Architecture (MAA). The flow of control signals is from higher to lower priority agents. According to the distribution of the control between the agents, hierarchical architecture can be further classified as being a simple and uniform hierarchy.

Holonic

The holonic agent organization is a stable structure that consists of several “holons” as its sub-structure and is itself a part of a larger framework.

According [10, 31] the hierarchical structure of the “holon” and its interactions have been used to model large organizational behaviors in manufacturing and business domains.

An agent that manifests as a single entity in a holonic multi-agent system may be made up of several sub-agents connected by bonds. The sub-agents are constrained by promises rather than by strict guidelines or predetermined regulations. These are the relationships that all of the agents inside the holon have consented to.

Every holon chooses a Head Agent who can interact with other agents in the surrounding area. The internal architecture of each agent, communication ability, and resource availability are typically taken into consideration while choosing the head agent.

In a homogeneous Multi-Agent System, the selection can be random, in the heterogeneous architecture, the selection is based on the capability of the agent. The holons formed may group further by benefits foreseen in forming a coherent structure.

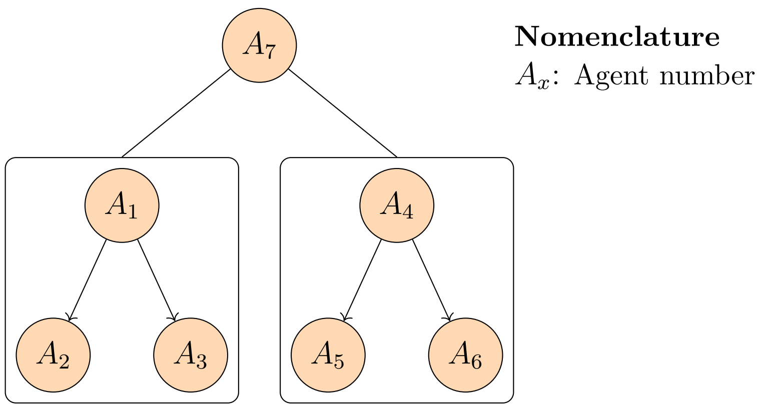


Figure 2.13: Example of Superholon resembling the Hierarchical MAS (Adapted from [10]).

Figure 2.13 shows a Superholon formed by grouping two holons. The agents A_1 and A_4 are the heads of the holons and communicate with agent A_7 this is the head of the Superholon. In holonic architecture, cross-tree interactions and overlapping agents forming part of two different holons are allowed.

Coalitions

In coalition architecture, a group of agents comes together for a short time to increase the utility or performance of the individual agents in a group. The coalition ceases to exist when the performance goal is achieved [10].

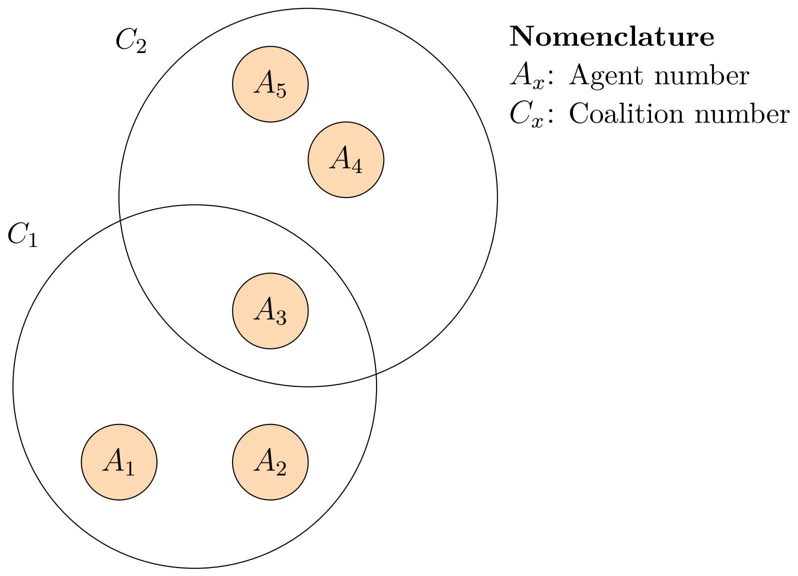


Figure 2.14: Coalition Multi-Agent Architecture (Adapted from [10]).

Figure 2.14 shows a typical coalition Multi-Agent System using overlapping groups (C_1 and C_2). The agents that form the coalition can have a flat or hierarchical architecture.

Agent overlap is permitted within coalition groups since it fosters a greater level of mutual understanding. Nevertheless, the strategy computation becomes more difficult when overlap is used.

In theory, the system's performance will be maximized if all of the agents in the environment unite into a single group. This is because every agent has access to all the data and tools required to determine the ideal course of action.

Reducing the number of coalition groups formed is necessary to lower the expenses related to forming and disbanding a coalition group. The creation of the group could happen live or it could be predetermined based on a cutoff point for performance metrics.

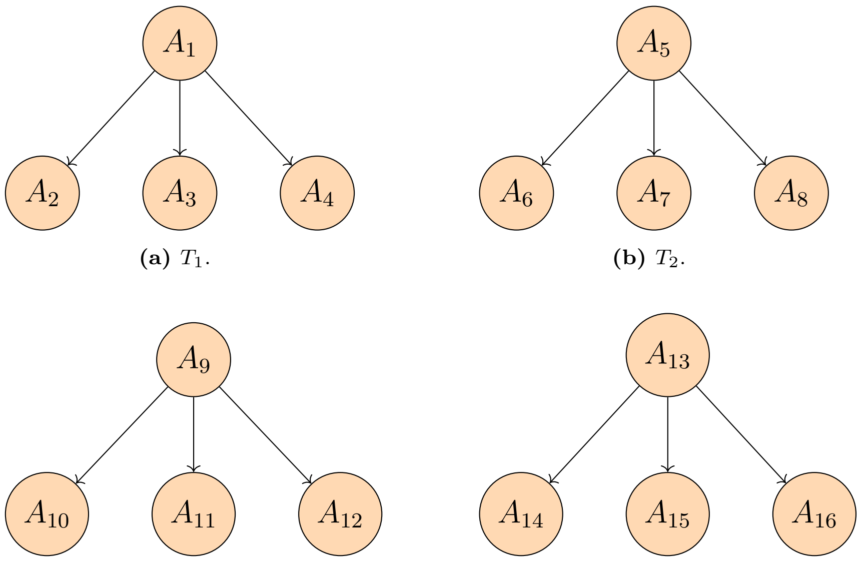
Teams

According to [32] Team MAS architecture is similar to coalition architecture in design except that the agents in a team work together to increase the overall performance of the group.

According [10] the paper [33] deals with a team-based MAA having a partially observable environment. In other words, teams that cannot communicate with each other have been proposed for the Arthur's bar problem. Each team decides on whether to attend a bar using predictions based on the previous behavioral pattern and the crowd level experienced, which is the reward or the utility received associated with the specific time. Based on the observations, it can be concluded that a large team size is not beneficial under all conditions.

Large teams provide access to more pertinent information and a greater view of the surroundings. However, it has an impact on how one learns from or integrates the experiences of individual agents into a single framework team. Smaller teams can learn more quickly, but because of their narrower field of vision, they do less well than ideal. When choosing the ideal team size, trade-offs between performance and learning must be considered. When compared to coalition multi-agent system architecture, this results in a substantially higher computational cost.

A team based on a partial view architecture is shown in Figure 2.15. Team 1 (T_1) and Team 3 (T_3) can see each other, but not Team 2 (T_2) and Team 4 (T_4), and vice versa. Even in homogeneous environments, the agents' internal behavior and responsibilities are random. agent structure.



(a) T_1. **(b)** T_2. **(c)** T_3. **(d)** T_4.

Figure 2.15: Team Based MAA with a Partial View of the other Teams (Adapted from [10]).

2.4.3 Multi-Agent Learning (MAL)

Learning in MAS is very important, a system capable of learning and changing its way of acting provides great potential to face many problems for which don't know the behavior of other agents in the environment. This adds more levels of difficulty in tasks of coordination between agents.

According to [34] Multi-Agent Learning (MAL) allows designing certain guidelines from which an agent will be able to exploit the dynamics of its environment and use them to adapt to it. In a MAE, learning is more important and more difficult, since the selection of actions has to be done in the presence of other agents, who do not necessarily have to follow the rules of the environment and can make non-deterministic decisions. These agents in turn will adapt their actions to those previously carried out by the other agents.

According to the book [10] MAL can be classified into Active Learning, Reactive Learning, and Learning based on consequence. In Active and Reactive Learning, the update of the belief part of the agent is given preference over an optimal action selection strategy, as a better belief model could increase the probability of the selection of an appropriate action.

2.5 MAS Applications in Industry

MAS is increasingly recognized as a powerful tool in various industrial applications due to its ability to deal with complex, dynamic, and distributed environments. The book [35] includes papers and case studies on Flexible Automation and Intelligent Manufacturing which states that the industry's need for systems that can operate autonomously, coordinate tasks, and adapt to changing conditions is critical. The best way to meet these requirements is to apply MAS, as they have several advantages that make them ideal for solving complex and dynamic problems

In industry, MAS can be applied in manufacturing, supply chain management, energy management, transportation and logistics, and healthcare. In the manufacturing industry, MAS is widely used in intelligent manufacturing systems, enabling real-time monitoring, adaptive production processes, and the coordination of robotic teams. Agents can optimize production lines, manage supply chains, and ensure efficient allocation of resources.

A MAS can be applied to a simple case study, as well as an overly complex one. The paper [36] explains that it is possible to develop a MAS to achieve the best comfort preferences using low-cost hardware such as Raspberry, and at the same time improve consumption performance using one principal agent who will represent local system take into account any directives that may exist for this environment.

Regarding CPS and Industry 4.0, the paper [37] implements a MAS to transfer intelligence between the modular conveyors of Fischertechnik$^{\text{TM}}$ running on Raspberry Pi, supporting self-organization capabilities. Each device is considered an agent and incorporates a simple and distributed self-organization mechanism, composed of several simple rules, to control the system's operation in response to changing conditions [37]. In other words, this paper aims to develop a MAS so that the agents can perceive that the environment has changed and self-organize it.

In MAS there are several tools used in the areas of modeling and simulation, each with its focus and specific characteristics. JADE is a platform for MAS development, while NetLogo is an agent-based simulation platform aimed at teaching and research and has its script language. The paper [38] addresses the simulation of agent-based manufacturing systems using the Agent-Based Modelling (ABM) tool which offers a means of modeling and simulating complex adaptive systems through the use of agents and the results of the simulation can be produced graphically [38]. In this paper, the ABM NetLogo environment will be used to model and simulate an agent-based control system for a washing machine production line.

Chapter 3

Multi-Agent Platform

This chapter provides a practical overview, focusing on AOP and JADE. Keywords like AMS and DF are highlighted. Additionally, exchanging messages between agents using IMTP and FIPA, and communication protocols are briefly explained.

3.1 Agent-Oriented Programming

Before going into details about Multi-Agent Platform (MAP), it's important to mention the type of programming language used in this dissertation. For the development of MAS, there are several frameworks, one of them is JADE which will be used to develop a case study.

JADE is implemented in Java, therefore this section aims to explain what the Object-Oriented Language (OOL) and architecture of AOP consist of. OOL is a high-level computer programming language that implements objects and their associated procedures within the programming context to create software programs.

According to [8] agents have been considered as a function, receiving input from the environment (percepts) and returning output (actions). There are four classic ways to model the internals of agents: Logic Based Agents, Reactive Agents, BDI, and layered architectures. The main focus will be on the BDI model, as this is the leading approach to AOP. However, as the BDI model builds on top of the previous approaches, these will also be mentioned to give the full picture of AOP.

The traditional approach to AOP was to use logic in decision-making. This can be done by representing the world through a symbolic representation of the environment and the agent's goals. Then the representation can be manipulated until the goal state is achieved. Each predicate would be some belief about the world, for example, the agent's location or room temperature [1].

3.1.1 Belief-Desire-Intention

According to [8, 23] the BDI architecture is an attempt at modeling practical reasoning. It builds upon the logical and reactive approaches, which by themselves both had some faults. It tries to balance the proactive (goal-directed) and reactive (event-driven) elements of reasoning, using local-based agents and reactive agents.

This is done by taking the simplicity of representing the world using logic, and combining this with rules from the reactive approach. This allows for a simple way of modeling the environment while being able to react quickly to sudden changes.

Figure 3.1 illustrates how the BDI architecture is linked together and the connections between the individual elements in the architecture.

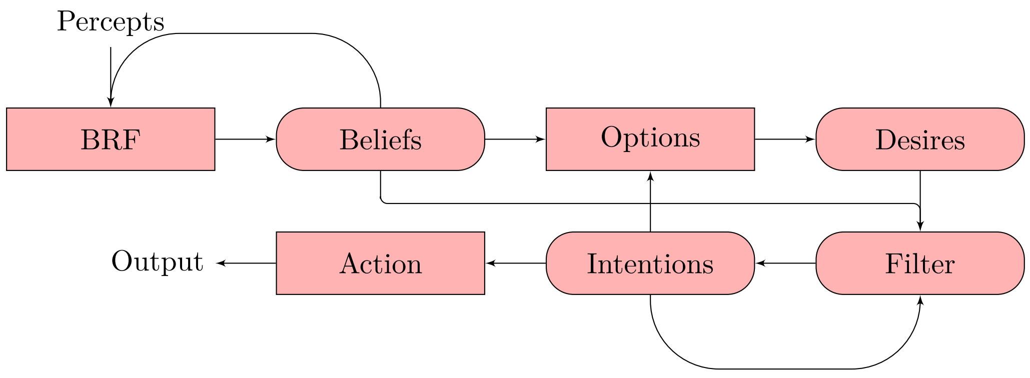


Figure 3.1: Overview of the BDI Model and its Functions (Adapted from [8]).

- **Beliefs:** The agent perceives the environment, receiving percepts and revising these using a Belief Revision Function (BRF). The BRF maps the percepts into new beliefs, merging them with the old ones and removing incorrect or outdated beliefs as a result [8].

- **Desires:** By evaluating the agent's current beliefs, together with its current intentions, an option-generating function is used to determine which options are available to the agent. These options represent the desires the agent wants to accomplish. An agent can commit to fulfill one or more of its desires, thus becoming intentions [8].

- **Intentions:** Is an actual goal, which the agent can attempt to achieve. This is done by finding relevant plans, being the recipes on solve goals. Each plan comprises a sequence of actions and a specific goal achieved by following the plan. As a result, goals can be solved by creating compositions of several plans [8].

- **Deliberation:** Utilizing a filter function, the agent can decide which desires to commit to. The function takes all the agents current beliefs, desires, and intentions into account, evaluating which of the intentions to proceed with. It should be able to act towards an intention continuously, but at the same time, dynamically change which intention to pursue. If the intention is accomplished, becomes impossible to achieve, or if the intention is no longer a desire, it should be dropped. This is the biggest challenge with the BDI architecture, finding the perfect balance between continuing with and reevaluating intentions [8].

The agent wastes resources that could be used to carry out the intentions if it reviews them too frequently. However, if the agent never reassesses its goals, it can wind up pursuing objectives that, in light of the environment as it stands now, are either irrelevant or more effectively accomplished. The dynamics of the environment and the agent's need for reaction determine how frequently an agent should reassess its goals [8].

Choosing the action taken to accomplish its current aim is the action function, which is the final stage in the BDI architecture. Plans are chosen based on their indicated prerequisites, and actions are chosen based on the plans that are currently accessible. The agent then applies the specified action to the surroundings via its actuators.

Following the perception of the altered state of the environment, the agents' amended beliefs, desires, and intentions trigger a new round of deliberation. Up till no more urges are felt, this process will be carried out.

3.2 JADE Platform

JADE is a platform that facilitates the construction of distributed systems providing an abstraction of software agents over the Java programming language.

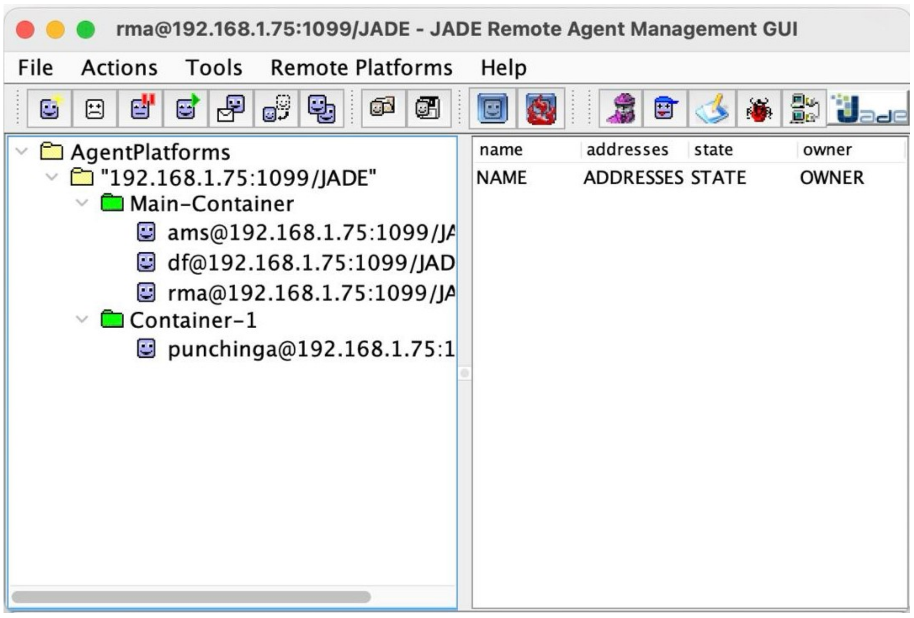
JADE is known for its flexibility, scalability, and compliance with FIPA standards. In addition, it offers robust support for agent communication, agent lifecycle management, and service discovery. The framework is highly modular, allowing complex systems to be created from smaller, reusable components. One of the main aspects of JADE is the organization of agents in containers.

A container is a Java Process providing the JADE run-time and all the service needed for hosting and executing agents, and agents live in containers. The main container is a special container representing the bootstrap point of the platform. All the containers must join to a main container by registering with it. By default, according to [1] the main container contains three agents:

- Agent Management System (AMS) that supervises the entire platform;
- Directory Facilitator (DF) implements a set of static methods to communicate with a standard FIPA DF service that means DF implements the Yellow Pages Service;
- Remote Monitoring Agent (RMA) is a system tool that implements a graphical platform management console.

According to the book [1] JADE is a framework that facilitates the creation of distributed applications that take advantage of the software agent abstraction by offering fundamental middleware-layer features that are independent of the particular application. The fact that JADE implements this abstraction over the popular OOL Java and offers a user-friendly and straightforward Application Programming Interface (API) is one of its key advantages.

Figure 3.2 illustrates the JADE RMA Graphical User Interface (GUI) where it shows the status of Agent Platform (AP) is required to provide main container with the three agents discussed above.


rma@192.168.1.75:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help

```
AgentPlatforms
  "192.168.1.75:1099/JADE"
    Main-Container
      ams@192.168.1.75:1099/JA
      df@192.168.1.75:1099/JAD
      rma@192.168.1.75:1099/JA
    Container-1
      punchinga@192.168.1.75:1
```

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

Figure 3.2: Screenshot of the JADE RMA GUI.

For this to work, it's necessary to use the `-gui` command. Without closing the GUI, using for example, `-container:name.nameclass`, it's possible to view the container.

3.2.1 JADE Architecture

According to [1] the core of the JADE system is ACL, which contains seven layers: transport, encoding, messaging, ontology, content expression, communication, and Internet Protocol (IP).

AMS is additionally part of a system for ACL and provides physical infrastructure, in which agents exist. AMS is a crucial and mandatory component of AP. It manages the operation of AP and enables the basic agent life cycle by controlling or identifying. In JADE controls the platform's use, and it's responsible for agent control and registrations.

DF which is an optional component to the AP, provides information about the agent residing in current. If an agent wishes to make its services public, it has to register in DF, this is also called a yellow pages service.

The yellow pages service facilitates the registration and discovery of services offered by agents within a MAS. Regarding registration agents can register themselves with the DF, specifying the services they offer along with other relevant information.

Regarding discovery, other agents can query the DF to find the services they require. The DF then returns a list of agents that offer the requested service, along with their contact information and other details. By providing this service, the DF enables agents to dynamically discover and interact with other agents that offer specific services, promoting flexibility and coordination within the MAS.

Message Transport Service (MTS) is a member of AP, which is responsible for transporting agent messages. Agent Communication Channel (ACC) is an agent, which is responsible for the outside-to-inside platform communication between agents. It offers reliable and accurate message routing and supports Internet Inter-ORB Protocol (IIOP) for interoperability between different AP.

Figure 3.3 shows JADE architecture, according to [1] is built upon AP, containers, and main container.

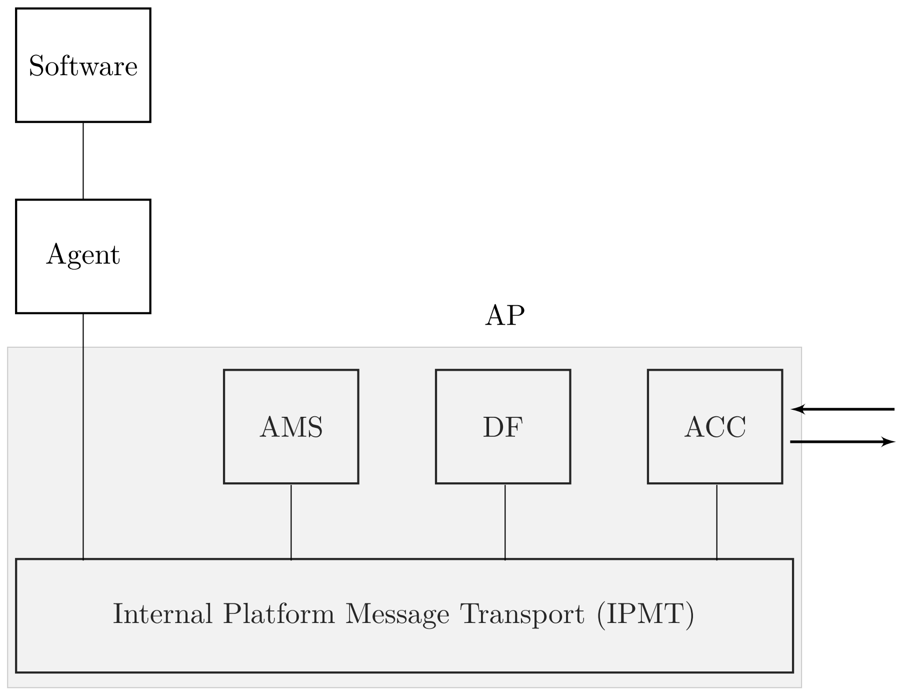


Figure 3.3: FIPA Reference Model of an AP [1].

In each platform, there is one host designated, which holds the main container.

In this container, the main agents are residing, which are necessary for running the whole platform. When the main container is created, other containers can join the platform.

The containers can be on different computers, but when registering, the system can locate them. Each container is a multithreaded execution, some local server, which allows maintaining agents.

According to [1, 39] in JADE there are several mechanisms introduced to raise the efficiency of the platforms, like using cooperative behavior to reduce the overall thread number or recycling JADE objects, which saves time by reducing the dynamic memory allocation.

3.2.2 Agents

In JADE agents are small, autonomous, and operate on a single thread. Each agent in the system has its distinct identifier. Each agent lives in a container and has a local name, which is unique in the whole system, and it's used to acquire the agent identifier to send the message to.

According to [1] agents are created using the `create` method in a specific container. The reference to an agent isn't visible to the programmer and other agents. So there is no possibility of invoking its methods by other agents. But an agent can create other agents, self-destruct or kill other agents, migrate, send messages, disable the container, and also shut down the entire platform if necessary.

Agent, lifecycle is covered by its methods. Just after birth or after migration it can add a behavior, create other agents, or set some values, like a stream object with access to the file.

The message communication is done using buffers and an agent can handle one message at a time. Of course, messages are sent asynchronously. When sending a message an agent doesn't have to know, where the receiver is, in which container. It might even not exist it only has to know its identifier and the system will find it. The only problem is that with a message there is only one object attached to it. It can be a character string but also any other object. An agent can then recognize it only by the message descriptors or by message performative.

Agent execution is covered in JADE by the sophisticated behavior class family. There are already many patterns of behaviors defined, which still execute using on agent thread.

JADE provides mobility between containers because these groups of agents performing one task can be distributed around many containers and still achieve a common goal.

3.2.3 Agent Model in JADE

According to [1, 40] agents in JADE are developed as Java classes that extend the `jade.core.Agent` class. They are created, have a run-time, then they are terminated.

Agents have predefined methods that correspond to these life cycle periods. Agents can override the setup method defined in `jade.core.Agent` class to have custom initialization steps [1].

Likewise, agents can override the `takeDown` method to have a custom finalization code. The basic structure for the lifecycle of JADE agents is given in Figure 3.4.

Each agent in a JADE platform must have a globally unique name. In practice, global uniqueness is achieved by (a process managed by the AMS) adding the platform name to the original name of the agent. The identification of each agent is stored in an instance of the `jade.core.AID` class [1].

3.2.4 Agent Activities

According to [40] actions an agent can do in a JADE platform are defined through behaviors. The behaviors that an agent can have during its run-time are defined by extending one of the behavior classes defined in the `jade.core.behaviours` package and add instances of these classes to the task queue of the agent.

According to [1] the `jade.core.behaviours.Behaviour` base class defines two public abstract methods that must be overridden by child classes.

- An action method (method signature: `public void action()`) defines the code that constitutes the task carried out by this behavior.

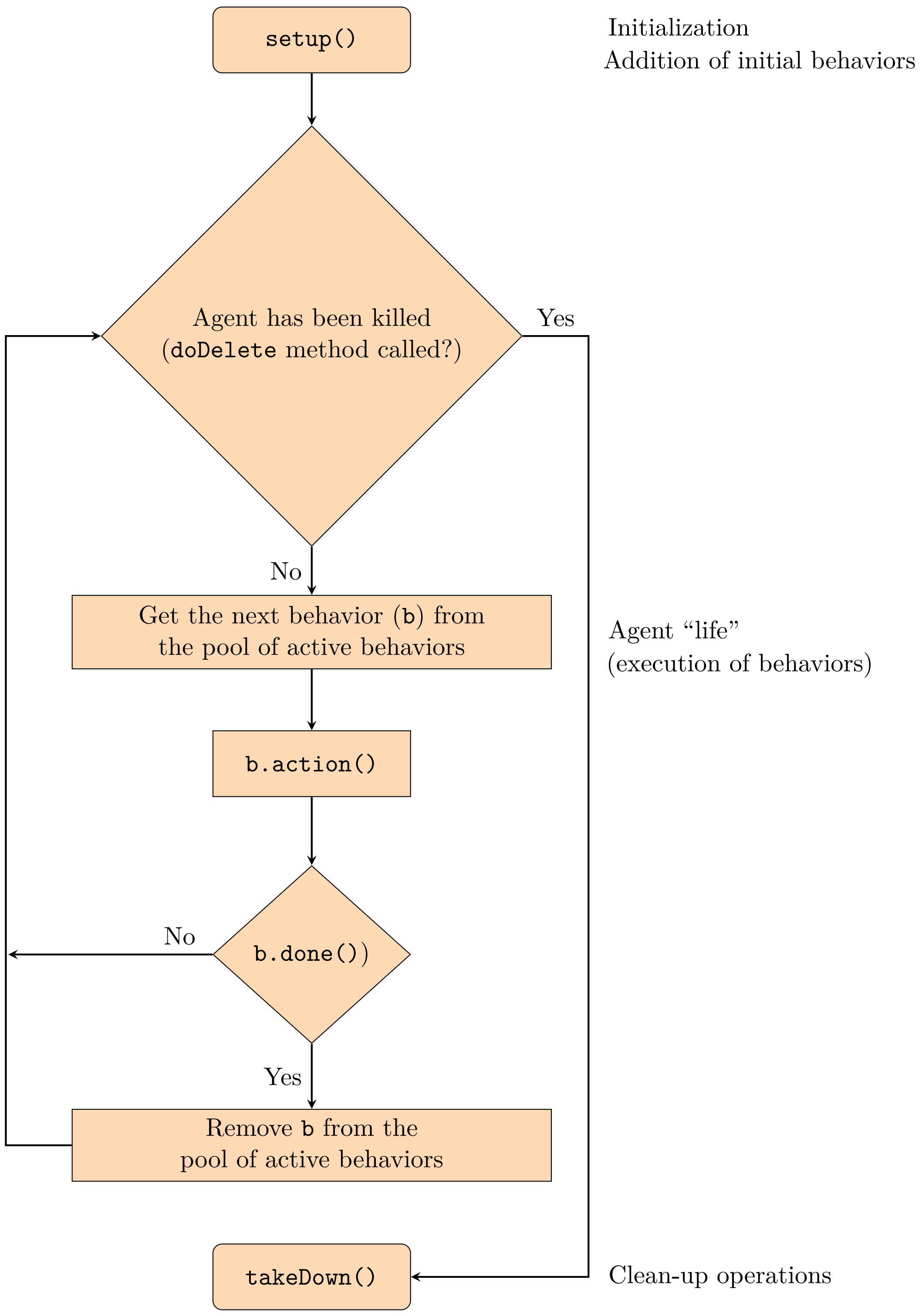


Figure 3.4: Agent Thread Path of Execution [1].

- A done method (method signature: `public boolean done()`) defines a control structure for checking the completion status of this behavior. As it can be seen from Figure 3.4 the done method is called after each execution of a behavior.

In addition to the above methods `jade.core.behaviours.Behaviour` class also defines two methods that can be overridden for further behavior customization:

- the `onStart()` method which is executed once before the first call to the action method (similar to the `setup()` method of `jade.core.Agent class`).
- the `onEnd()` method which is executed right after the `done()` method returns true (similar to the `takeDown()` method of `jade.core.Agent class`).

The Agent-core of each agent (provided by the superclass `jade.core.Agent`) keeps a task list for active behaviors [1].

3.3 Communication among Agents

Agents in JADE communicate with one another via writing messages that follow the ACL standard. This section's goal is to provide a brief description of the FIPA Agent Communication Language Specifications (FIPA-ACL) protocol, which permits agent-to-agent communication.

3.3.1 FIPA

FIPA-ACL deal with ACL, message exchange interaction protocols, speech act theory-based communicative acts, and content language representations. FIPA-ACL works with 20 communication acts divisible into 5 groups: information transmission, information request, negotiation, action realization, and error message.

According to [1] FIPA was established in 1996 to develop a collection of standards relating to software agent technology. FIPA objectives are as follows:

- Agent technology provide a new paradigm to solve old and new problems;
- To be of use, some agent technology requires standardization;

- Standardization of generic technologies is possible and to provide effective;
- The standardization of the internal mechanics of agents themselves is not the primary concern, but rather the infrastructure and language required for open interoperation.

Since 1996, FIPA has always changed to improve its methodology and new skills, for this purpose, it has created standards such as FIPA' 97, FIPA'98, and FIPA2000. As described in [1] to date, some key achievements of FIPA are as follows:

- A set of standard specifications supporting Inter-Agent Communication (IAC) and key middleware services;
- An architecture providing an encompassing view across the entire standards. This architecture underwent an incomplete reification as a Java Agent Services (JAS);
- A well-specified and much-used ACL, accompanied by a selection of content languages and a set of key interaction protocols;
- Several open-source and commercial agent tool-kits with JADE.

According to [41] the FIPA-ACL message contains a set of parameters, which will vary according to the situation and which will consequently be necessary for an effective communication by the agents.

In general, the message structure will be defined through parameters that indicate the type of communicative act, the content, the participants involved (sender and receiver), the communication control and message description. Table 3.1 shows the parameters of the FIPA-ACL message.

Regarding the security of the FIPA-ACL protocol, this specification allows security management to be implemented in the message transport layer, through the use of security services available in a shared transport protocol on the AP.

According to [42, 43] FIPA provides a set of standards and specifications for agent communication in distributed systems. Some commonly used FIPA communication protocols in JADE include:

Table 3.1: ACL Message Parameters [1].

Parameter	Description
`performative`	Is a required parameter that defines the communicative act (or purpose) of the message. Some performative types are: `INFORM`, `PROPOSE`, `CFP` (Call for Proposals), `REQUEST`, `ACCEPT_PROPOSAL`.
`sender`	Gives the identity of the sender of the message (for JADE the Agent Identifier (AID)) and can be omitted if the sender desires to remain anonymous.
`receiver`	Identity of the intended recipients of the message
`reply-to`	Indicates that subsequent messages in this conversation thread are to be directed to the agent identified by the reply-to parameter.
`content`	Content of the message
`language`	Language in which the content parameter is expressed
`encoding`	Specific encoding of the message content
`ontology`	Reference to an ontology to give meaning to symbols in the message content
`protocol`	Interaction protocol used to structure a conversation
`conversation-id`	An expression used for identifying an ongoing sequence of communication acts (a conversation) between agents
`reply-with`	An expression to be used by a responding agent to identify the message
`in-reply-to`	Reference to an earlier action to which the message is a reply
`reply-by`	Specifies the latest time a reply is expected for this message, used for timeout definitions

- **FIPA Request Interaction Protocol:** This protocol is used when an agent needs another agent to perform a task or provide some information.
- **FIPA Query Interaction Protocol:** This protocol is used when an agent needs to query another agent for information. The querying agent sends a query message to the target agent, which then processes the query and sends a response containing the requested information back to the querying agent.
- **FIPA Contract Net Interaction Protocol:** This protocol is used when an agent needs to delegate a task to other agents and select the best agent to perform the task. The contracting agent sends a call for proposals to potential agents, who then submit proposals detailing how they would complete the task. The contracting agent selects the best proposal and assigns the task to that agent.
- **FIPA Subscribe Interaction Protocol:** This protocol is used for subscription-based communication. An agent can subscribe to receive notifications or updates from another agent on specific events or changes.
- **FIPA Inform Interaction Protocol:** This protocol is used for one-way communication where an agent informs another agent about some event or information without expecting a response.

These protocols provide standardized ways for agents to interact with each other, enabling efficient communication and coordination in MAS like those developed using JADE.

3.3.2 Messaging

Summarizing what [1] described, FIPA standard also specifies the ACL. ACL is based on message passing with individual messages between them using Java Remote Method Invocation (RMI) mechanisms in Java Virtual Machine (JVM). ACL specifies the language by setting out the encoding, pragmatics, and semantics of messages. Of course, the technology for handling this is not given. Generally, the message should be encoded in a textual form. Within ACL there is a possibility of including formal semantics. The standard also specifies common forms of agent interaction protocols (patterns of agents talking to each other).

Message in JADE is constructed of envelope (transport information including a destination), payload (encoded message), message (message parameters), and content (message content) as illustrated in Figure 3.5.

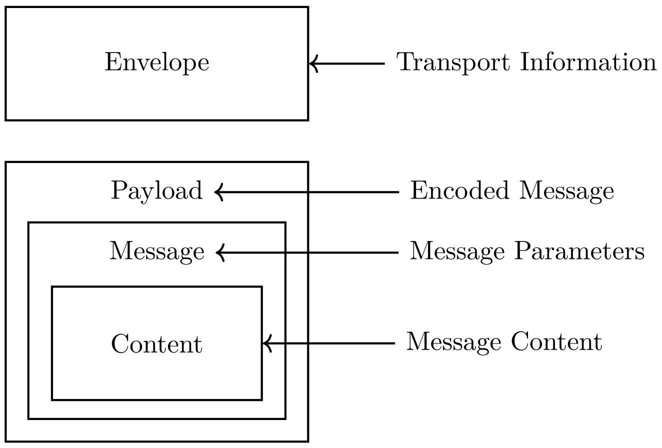


Figure 3.5: FIPA Message Structure [1].

These parameters make every message particular. For example, each message has its ontology, agent to reply to, or protocol used to structure a conversation. Moreover, commutative acts are strictly divided by their function: passing information, requesting information, negotiation, performing actions, and error handling.

In JADE there is also a possibility to communicate with a different AP, which is not a JADE one. IIOP is a standard protocol used for communication between distributed objects in an Common Object Request Broker Architecture (CORBA) environment.

According to book [1] to promote interoperability between different platforms, JADE implements all the standard Message Transport Protocol (MTP) defined by FIPA, where each MTP includes the definition of a transport protocol and a standard encoding of the message envelope.

By default, JADE always starts a Hyper Text Transfer Protocol (HTTP) based MTP with the initialization of a main container. MTP is activated on normal containers. In effect, this creates a server socket on the main container host and listens for incoming connections over HTTP at the Uniform Resource Locator (URL) specified. Whenever an incoming connection is established and a valid message is received over that connection, the MTP routes the message to its final destination which, in general, is one of the agents located within the distributed platform. Internally, the platform uses a proprietary transport protocol called IMTP which is described in 3.3.4.

JADE performs message routing for both incoming and outgoing messages using a single-hop routing table that requires direct IP visibility among containers. Using command-line options, any number of MTPs can be activated on any JADE container, including MTPs implementing different transport protocols. MTP can also be "plugged in" and instantiated at run-time by exploiting the RMA GUI console. This allows the platform the freedom to manipulate topology.

When a new MTP is activated on a container, the entire JADE platform gains a new transport address, a new endpoint where messages can be received.

3.3.3 Content Languages and Ontologies

According to [1, 40] in realistic scenarios agents in a MAS often need to communicate complex information (such as database records, actual Java objects, etc.). Content languages define syntaxes for representing complex data in ACL message slots. An example of content language is the FIPA defined Semantic Language (SL) which represents data as a string sequence. The conversion from complex data representation to a content language syntax is done via special codecs that handle the encoding and decoding processes.

An ontology defines mappings (name, type, and validation) between named items in a ACL message content and actual fields and methods inside a Java class. Thus using an ontology processing decoded ACL message content into a usable Java class is automatized. When used together, content languages and ontologies provide a simple, effective, and truly platform-independent framework for serializing and deserializing agent communication content.

3.3.4 IMTP

According to the book [1] JADE IMTP is exclusively used for exchanging messages between agents living in different containers of the same platform. It's considerably different from inter-platform MTP, such as HTTP. First, as it's used for internal platform communication only, it does not need to be compatible with any FIPA standards. It can be proprietary and thus designed to enhance platform performance.

The JADE IMTP is used not only to transport messages but also to transport internal commands needed to manage the distributed platform, as monitor the status of remote containers. For instance, it's used to transport a command to order the shut-down of a container, as for monitoring when a container is killed or becomes otherwise unreachable.

Figure 3.6 shows the main architectural elements of a JADE platform, but it's possible to see that it's possible to exchange messages between agentss living in different containers via IMTP as described above.

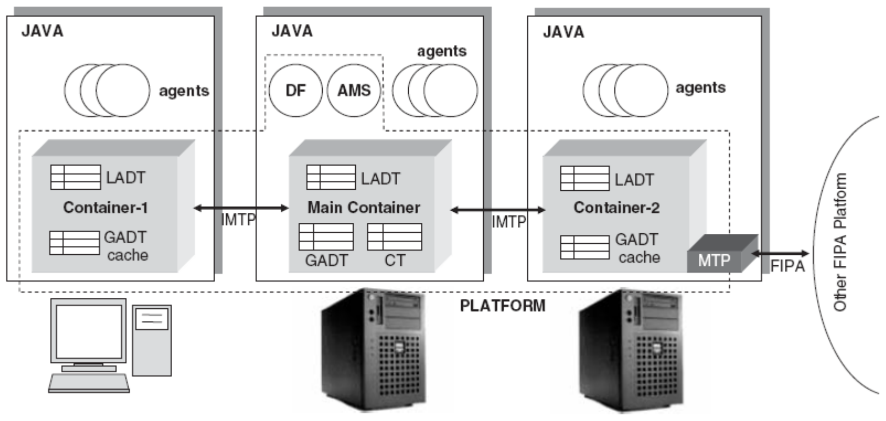


Figure 3.6: Relationship between Main Architectural Element [1].

3.4 Communication Protocols

This section aims to describe some communication protocols that are usually used in the interconnection of physical and cyber parts in a CPS perspective such as Message Queuing Telemetry Transport (MQTT), MODBUS which is normally used in IoT applications, and OPC Unified Architecture (OPC-UA).

3.4.1 MQTT

According to [44] MQTT is publishing and subscribe message protocol, being considered extremely simple and lightweight, designed for restricted devices and networks with low bandwidth, high latency, or unreliable. In addition, it's ideal for implementation in low-power devices, such as micro-controllers, due to its small size, low power consumption, minimized data packages, and efficient distribution of information to one or several receivers, facilitating the use in remote sensors of the IoT.

In this protocol, there is a broker that is the entity responsible for managing the messages and, for the exchange of messages to be carried out, all clients must be connected to the broker. The exchange of messages is carried out by topics when a publisher publishes a message topic, the broker receives this message and forwards it to the subscriber who subscribed to the same topic, as shown in Figure 3.7.

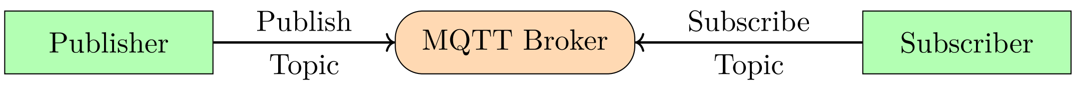


Figure 3.7: MQTT Behavior.

Regarding MQTT security support, this protocol by default isn't secure. However, it's possible to pass a username and password with a MQTT package, and encryption on the network must be handled with Secure Sockets Layer (SSL), but this can add significant overhead on the network since SSL isn't a light protocol. In addition, additional security can be added by an application that encrypts the data it sends and receives, but this doesn't keep the protocol simple and lightweight [44].

In MQTT Protocol have a parameter that defines the level of guarantee for message delivery between MQTT clients and brokers, designated by Quality of Service (QoS). There are three levels of QoS, the QoS 0 means best effort, the QoS 1 means at least once, and finally, the QoS 2 means highest level.

QoS 0 is suitable for scenarios where the loss of some messages is acceptable or high frequency, while QoS 1 ensures delivery at least once to the intended recipient.

However, this level can result in duplicate messages if acknowledgment is lost or a client republishes the same message due to a timeout.

In MQTT Explorer, it's possible to choose the appropriate QoS level depending on the reliability requirements of the application, network conditions, and the acceptable level of overhead. It's essential to balance between message reliability and the resources required for message transmission and processing.

3.4.2 MODBUS

According to [2, 11] in 1979 the company Modicon developed a serial communication protocol for their Programmable Logic Controllers (PLC). From that moment this protocol was expected as the unofficial standard because of its simplicity and robustness. It offers communication between clients/servers on various networks or many buses, it even is despite its old age, it's still the standard today. MODBUS have three main versions of MODBUS which are as follows:

- MODBUS Remote Terminal Unit (RTU) means that for communicating over serial using binary representation of data;
- MODBUS ASCII means that is for communication over serial using ASCII characters for protocol communication;
- And MODBUS Transmission Control Protocol (TCP) means that is using TCP/IP for communication.

According to [2] MODBUS RTU and ASCII needs the slave ID at the beginning of the message and a checksum at the end. While MODBUS TCP needs only an MODBUS Application Header (MBAP Header) at the start of each message. The data part of a MODBUS message is called a Protocol Data Unit (PDU) and is the same for all three versions of MODBUS discussed previously.

The PDU consists of MODBUS function code and then data. The whole MODBUS message, where the header and PDU is combined is called an Application Data Unit (ADU). A comparison of the ADU structure for MODBUS RTU and MODBUS TCP can be seen in Figure 3.8.

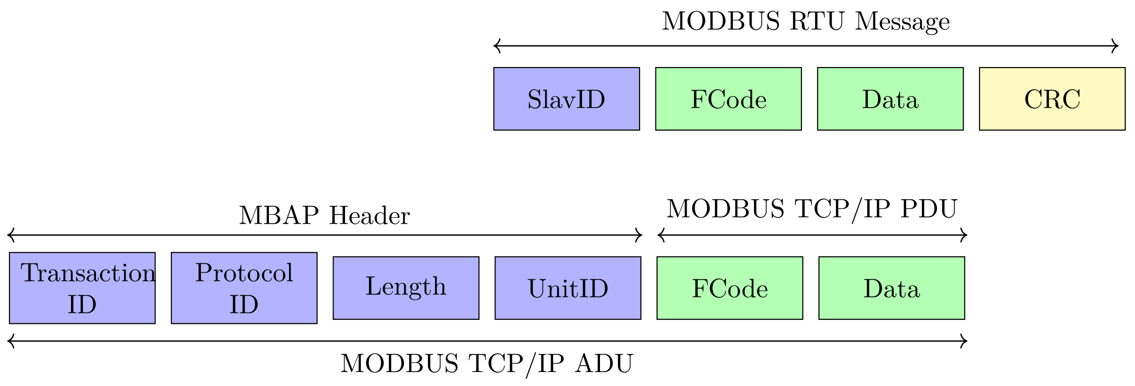


Figure 3.8: MODBUS RTU ADU and MODBUS TCP ADU [11].

According to book [11] in the area of industrial communication protocols, the industry-standard MODBUS can connect industrial electronic devices. The main reasons why MODBUS is used more widely than other communication protocols are:

- Published and no copyright requirements;
- Easy to deploy and maintain;
- There aren't many restrictions for the vendor to modify the mobile local bits or bytes.

MODBUS allows multiple devices to be connected to communicate on the same network, for example, a device that measures temperature and humidity and sends the results to a computer. In the Data Acquisition and Monitoring Control System (SCADA), MODBUS is typically used to connect monitoring computers to RTU systems.

According to [2] the function code is a set of codes specified by the MODBUS specification. Storing the data in MODBUS is done on the slave in four different registers. An overview and specification for each register are shown in Table 3.2.

When it comes to security the MODBUS protocol has nothing to offer. And since it has a known data structure for sending data, the data can easily be intercepted and read by unwanted persons. In addition, when using MODBUS TCP, anyone who gets access to the network can connect to the device and send commands. The same applies for MODBUS RTU and MODBUS ASCII.

Table 3.2: MODBUS Registers Overview [2].

Register	Access	Size
Coil	Read/Write	1 bit
Discrete	Input Read Only	1 bit
Holding	Read/Write	16 bit
Input	Read Only	16 bit

3.4.3 OPC-UA

The OPC-UA is already a mature industrial communications specification and there are many good introductions to the subject [12, 13].

OPC-UA is essentially an application layer protocol in industrial and automation use cases. The main purpose of an application layer protocol is to define how processes running on different end systems pass messages to each other. An application layer protocol defines the types of messages (requests and responses), the syntax of the messages (fields), the meaning of the information in those fields, and the rules that determine when and how a process sends messages and responds to messages [45]. Industrial computer systems' control layer forms are made to send information to higher levels, such SCADA, as well as to exchange data with one another in real-time. OPC-UA is an example of a pattern designed to handle this type of communication. The fundamental parts of OPC-UA are data transport and information modeling [12].

OPC-UA protocol is an example of the client-server communication model. All communication between OPC-UA applications is based on the exchange of messages initiated by the client application. Usually the OPC-UA server resides on an automation device. One of the areas where OPC-UA has excelled is PLC interfaces. A OPC-UA server encapsulates the source of process information like a PLC and makes the information available via its interface. A OPC-UA client, for example, SCADA system, connects to the OPC-UA server and can access and consume the offered data. Applications consuming and providing data can be both client and server at the same time. OPC-UA applications mostly use binary encoding and

TCP transport protocol called UA TCP to access automation data. Also, OPC-UA communication is inherently tightly coupled and session-oriented.

OPC-UA standard consists of 24 parts listed in [46]. The specification describes UA's internal mechanisms, which are handled through the communication stack and are mostly only of interest to those that port a stack to a specific target or those that want to implement their own UA stack. OPC-UA services are defined in specification part 4 and organized into nine service sets as illustrated in Table 3.3.

Table 3.3: OPC-UA Service Sets [3].

Service set	**Use case**
Discovery	Discover servers and their security settings.
Secure channel	Services related to the security model.
Session	Maintain the session between a client and a server.
Node management	Modify the address space.
View	Browse through the address space.
Attribute	Read and write attributes of nodes.
Method	Call methods.
Monitored item	Setup monitoring for attribute value changes or events.
Subscription	Subscribe for attribute value changes or events.

OPC-UA standard is defined by the same abstract features that can be implemented with different protocols. This is done to increase the flexibility of the standard [12]. Specification part 6 describes the mapping between the security model.

The mapping specifies how to implement an OPC-UA feature with a specific technology. For example, the OPC-UA binary encoding is a mapping that specifies how to serialize OPC-UA data structures as sequences of bytes. Mappings can be organized into three groups: data encodings, security protocols, and transport protocol as illustrated in Table 3.4.

A subscription is a context to exchange data changes and event notifications between the server and client. Clients send publish requests to servers and receive

Table 3.4: OPC-UA Mappings [4].

Group	Options
Data encodings	UA Binary or UA XML
Security protocols	UA-SC
Transport protocols	UA-TCP or HTTP

notifications in the form of publish responses. The biggest benefit of the subscription model is efficiency compared to polling values periodically. A client can subscribe to three types of information from an OPC-UA server. These three types are variable value changes, event notifiers, and calculated aggregate values. A subscription is used to group sources of information together. A monitored item is used to manage a source of information. A piece of information is called a notification. A subscription can contain all three different types of monitored items. Figure 3.9 illustrates the association of session, subscription, and monitored item.

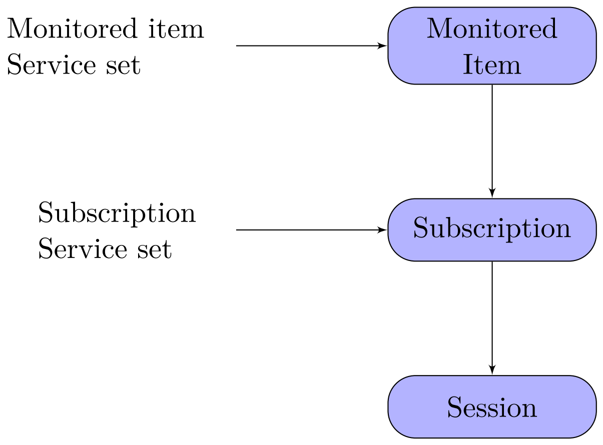


Figure 3.9: OPC-UA Subscription Model [12].

A subscription requires a session to transport the data to the client. The subscription can be transferred to another session, for example, to be used in a session created by a redundant backup client if the client that created the subscription is no longer available. Therefore, the subscription lifetime is independent of the session lifetime and a subscription has a timeout that gets reset every time data or keep-alive messages get sent to the client [12].

OPC-UA specifies security capabilities, which make use of standard public key cryptography mechanisms. If HTTP protocol is used, then TLS security is used to encrypt the traffic already in the Transport Layer. Figure 3.10 illustrates the concepts of the transport layer, secure channel, and session. This enables authentication of the applications which may communicate with each other.

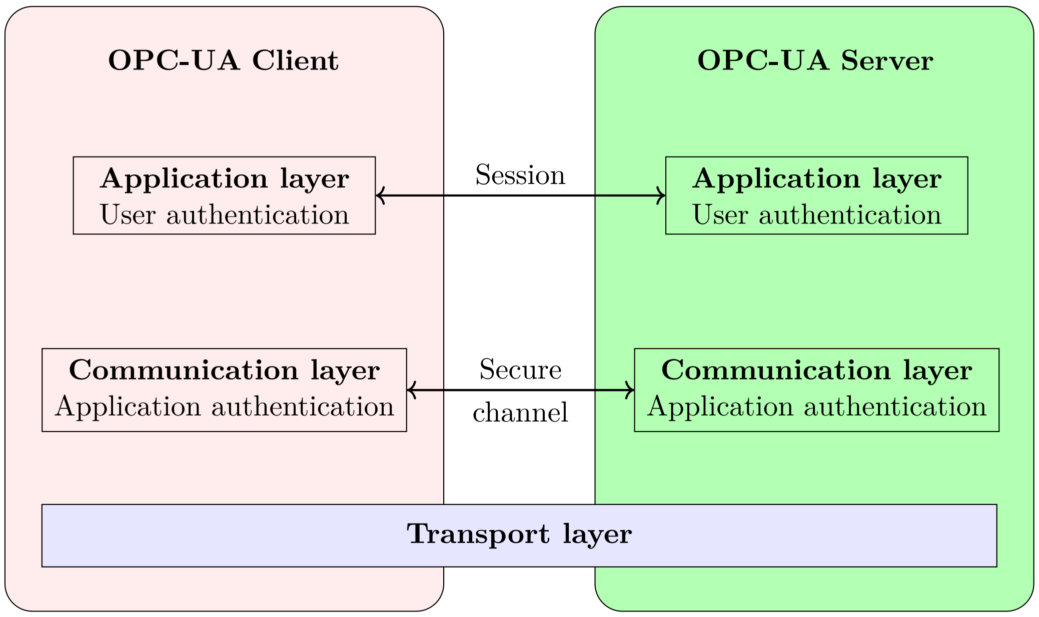


Figure 3.10: Transport Layer, Secure Channel, and Session [13].

Chapter 4

Small-Scale Production System

This chapter provides a case study where an MAS utilizing the JADE in the LCAR Manufacturing Cell.

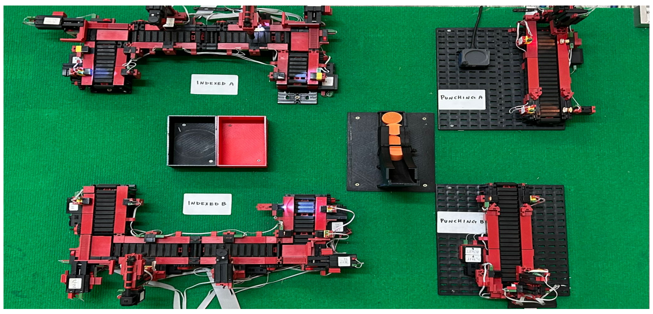
4.1 Case Study

System simulation can be defined as the imitation of the operation of a real-world process or system over time [47] using a model that represents the characteristics and functionalities of such processes or systems. Basically, during the simulation process, the model is exercised by manipulating the input parameters and analyzing how they affect the output performance indicators. Simulation is used to study, evaluate, test, and optimize complex systems from many domains.

The experimental case study considered in this dissertation is a real small-scale production system located at ESTiG LCAR composed of one IRB 1400 ABB robot (Figure 4.1a), two punching machines, and two indexed lines, supplied by Fischertechnik^{TM} (Figure 4.1b).

Fischertechnik^{TM} is a flexible and innovative construction system, built around the unique single Fischertechnik^{TM} building block, which allows connection on all six sides. The Fischertechnik^{TM} system of building blocks, motors, sensors, lights, computer controllers, and software controllers and software allows the user to develop countless models.

(a) IRB 1400 ABB Robot.



(b) Fischertechnik^{TM} Devices.

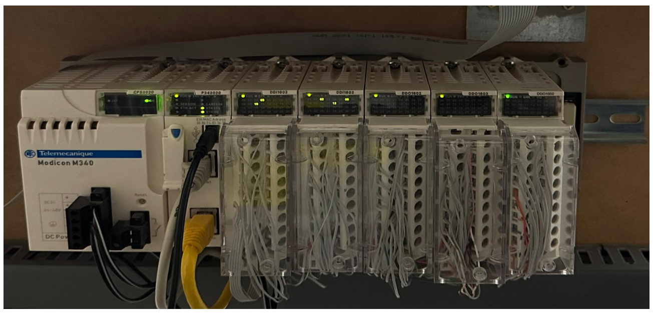
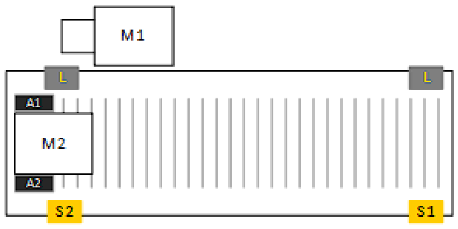
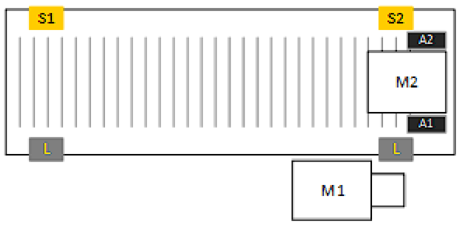

(c) Modicon M340 PLC.

Figure 4.1: Small-Scale Production Experimental Layout.

There are four Fischertechnik^{TM} lines required for this case study, two called Punching A (Figure 4.2a) and Punching B (Figure 4.2b) to simulate the punching of a part, and two called Indexed A (Figure 4.3a) and Indexed B (Figure 4.3b) to simulate the drill of a part.



(a) Punching A.

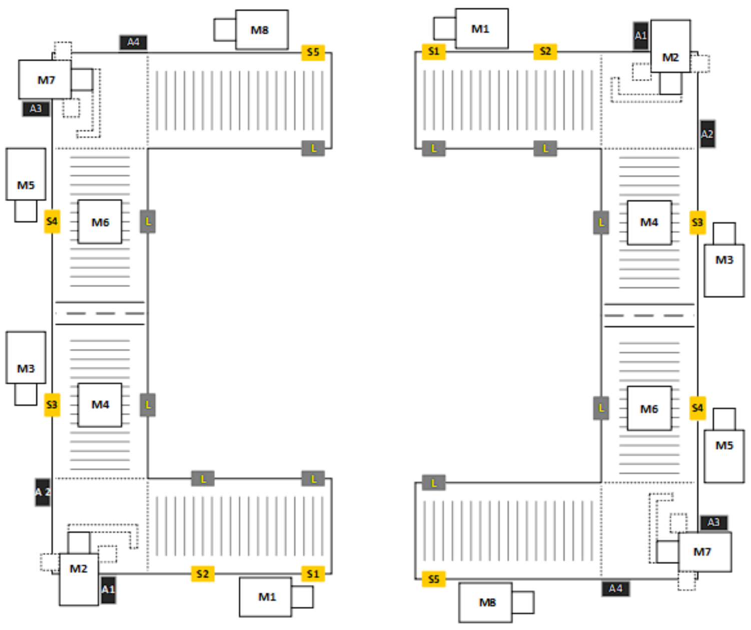


(b) Punching B.

Figure 4.2: Technologically Layout of Punching Machine.

Technologically, the punching machines are composed of two motors, a motor that moves a conveyor belt and another motor that moves the punch. As for the sensors, it has four, two light sensors to detect the products at the beginning of the conveyor and in the punching position, and two switch sensors to detect the end of the movement of the punching device as shown in Figure 4.2.

The indexed line is also a simulator of an assembly line, but this one is a bit



(a) Indexed A. **(b)** Indexed B.

Figure 4.3: Technologically Layout of Indexed Machine.

more complete than the punching machine. The indexed lines are composed of two workstations interconnected by several conveyors. Four switch sensors are used to determine the range of movement of the embolus and five light sensors are used to detect the presence of the products in the conveyors and the processing positions as shown in Figure 4.3.

The low-level logic control of the Fischertechnik^{TM} devices runs in a Modicon M340 PLC (Figure 4.1c) through proper IEC 61131-3 programs, which all the sensors on these lines were connected to the two input cards, as well as the actuators to the three output cards, as this PLC is modular.

Each line has the task of performing a certain task, which is designed to simulate a production environment for two types of products.

In the case of the punching machine (Figure 4.2), when sensor S1 detects the presence of the part, it activates motor M1, so that it can be transported on a conveyor belt, and when it reaches sensor S2, motor M2 will be activated to simulate working on the part. This part is returned by the same M1 motor until it reaches the S1 sensor.

In the case of indexed lines (Figure 4.3), the part will be transported by the mats activated by motors M1, M3, M5, and M8, it will be started when sensor S1 is activated; when passing sensor S2, a timer is activated to ensure that the part arrives at the correct place to be pushed by motor M2, to continue on the mat, when reaching the sensors S5 and S6, the work will be simulated by the corresponding motors M4 and M6, to finish the work, the part will be pushed again by motor M7 and will be directed by motor M8 to the final sensor S9.

The manipulator robot executes the transfer of products between the machines using proper RAPID programs. Finally, a human operator performs visual inspection operations to verify the parts' compliance with the specifications.

The dissertation aims to develop an MAS in JADE to control the production system. Each device will be considered as an agent. The agents communicate with each other to report their current status (free or occupied), actions, and behaviors through messages formatted according to the FIPA-ACL specification.

4.2 Agents

This section describes the agents developed in the JADE framework for the flexible small-scale production system mentioned above. The following agents were used to complete this system:

- Client Agent, who is responsible for requesting the human operator, the product, and the desired quantity;
- Product Agent, which is responsible for managing the production process of the catalog of products, are considered two agents of the Product Agent type because there are only two types of products, Product A and Product B;
- Resource Agent, which manages the execution of processing tasks by the resources disposed of in the assembly system, punching machines, indexed lines, inspection, and IRB 1400 ABB robot, six agents of the Resource Agent type are considered, corresponding to five of the devices discussed above.

Figure 4.4 shows the diagram of a small-scale production system. This diagram highlights the interaction between various agents developed using the JADE framework. This diagram also shows the communication flow between these agents using different types of FIPA-ACL. Additionally, the DF is included, which helps in locating the agents and managing their services. Interaction with the inspection agent is mediated through the Node-RED MQTT Broker, where topics are subscribed and published.

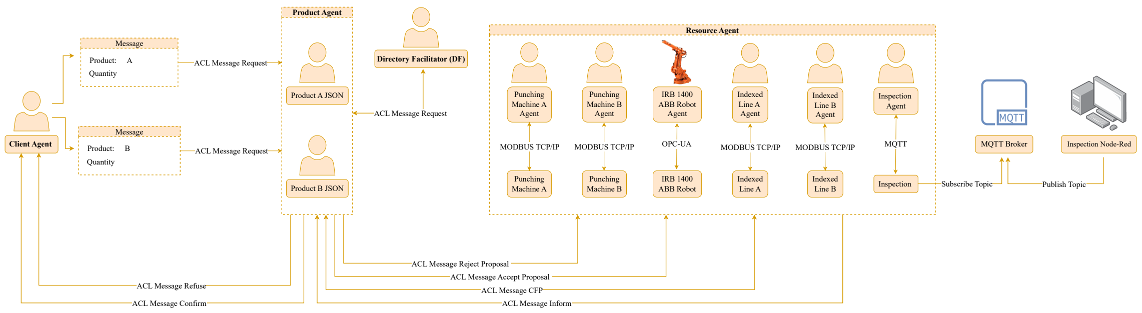


Figure 4.4: Small-Scale Production System Diagram.

In this case study, a total of ten agents will be used, as shown in Figure 4.4, the Client Agent which is responsible for requesting the production of a particular product, two agents of the Product Agent type which are responsible for managing the production process by negotiating with the Resource Agent, and six agents of the Resource Agent type which are responsible for carrying out manufacturing operations and finally the DF agent will be used as a resource to provide agents with a particular service. The behavior of each agent will be explained in detail below. The aim is for the agents to control the devices to fulfill the requests of the other agents and to be intelligent enough to realize that the environment has changed and to take decisions and actions under the environment.

4.2.1 Client Agent

The Client Agent behavior is to ask the human operator which of the products shown in Table 4.1 is desired. To do this, a GUI Form was developed as shown in the screenshot in Figure 4.5.

By clicking the Request button, the Client Agent sends a message according to the FIPA-ACL specification with the performance `REQUEST` to the Product Agent requesting a certain quantity of a product. In this case, the sender is the Client Agent and the receiver is the Product Agent.

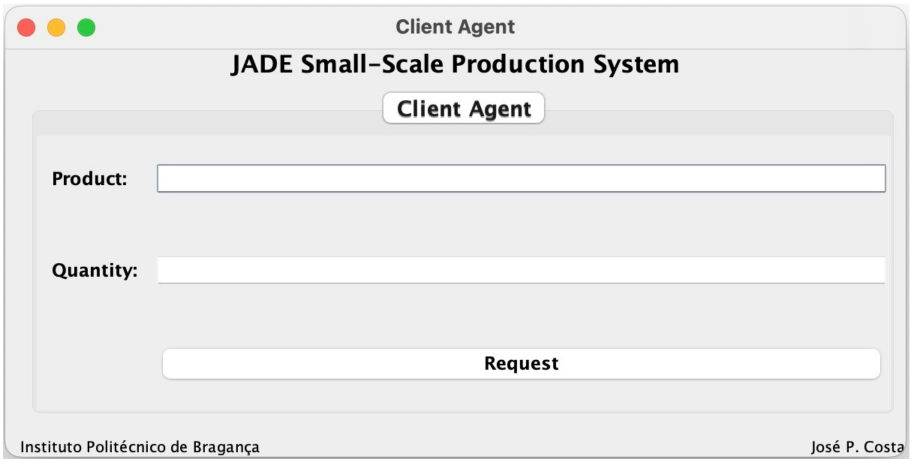

Client Agent
JADE Small-Scale Production System
Client Agent
Product:
Quantity:
Request
Instituto Politécnico de Bragança
José P. Costa

Figure 4.5: Screenshot of the Client Agent GUI Form.

When the Client Agent request is complete, it receives a reply message to inform it that the product has been completed with performance `CONFIRM`. If any problems occur during the production of the part, the Product Agent sends an ACL message to the Client Agent with the performance `REFUSE` as shown in Figure A.1. After receiving the Client Agent can then request another product. If the opposite is true: the sender is the Product Agent and the receiver is the Client Agent.

Listing 4.1, shows the Java code used for the communication method through FIPA-ACL between the Client Agent and the Product Agent.

Listing 4.1: Client Agent Behavior.

```java
addButton.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent ev) {
  try {
   ACLMessage acl = new ACLMessage(ACLMessage.REQUEST);
   if (product.equalsIgnoreCase("A")) {
    acl.setContent(product+":"+quantity);
    acl.addReceiver(new AID("ProductA", AID.ISLOCALNAME));
   } else if (product.equalsIgnoreCase("B")) {
    acl.setContent(product+":"+quantity);
    acl.addReceiver(new AID("ProductB", AID.ISLOCALNAME));
   }
   send(acl);
  } catch (Exception e) {
   logger.error(e.getMessage(), e);
  }
 }
});
```

In this dissertation, only one type of Client Agent will be considered, but in MAS it is always possible to consider several agent types of the Client Agent.

4.2.2 Product Agent

In this case study, two agents of the Product Agent type called Product A agent and Product B agent will be considered because two different types of products can circulate in the system, each one having a process plan, as illustrated in Table 4.1, which represents the sequence of operations to be performed.

Table 4.1: Representation of the Process Plans for the Product Agent.

Sequence	**{Product A}**	**{Product B}**
#1	{transfer_1}	{transfer_1}
#2	{punch}	{drill}
#3	{transfer_2}	{transfer_2}
#4	{drill}	{punch}
#5	{transfer_3}	{transfer_3}
#6	{inspect}	{inspect}
#7	{transfer_4}	{transfer_4}

The process plans will be stored in a JavaScript Object Notation (JSON) file to facilitate the logic of the Java Code, the purpose of which will be to configure the agents and place their sequences in each agent DF service.

The Product Agent behavior interacts with the Resource Agent through FIPA-ACL as the performance `CFP` to negotiate the price. To do this, the Resource Agent returns a message with performance `INFORM` and random price as content, and the Product Agent chooses the agent with the lowest price and sends `ACCEPT_PROPOSAL` ACL message, the agent with the highest price sends the `REJECT_PROPOSAL` ACL message as shown in Figure A.2.

To put the theory into practice, the DF agent will be asked several times throughout the process, because for the Product Agent to interact with the Resource Agent it needs to know the skills of the resources. In other words, the Product Agent asks

DF which agents have the service, once again using FIPA-ACL with performance `REQUEST` and with the content of the service it wants, and the DF agent sends a message `INFORM` with the agents' names as content, and the Product Agent sends a `CFP` message to request a service from the Resource Agent as shown in Figure A.2.

Listing 4.2 shows the Java method for the DF agent to search for agents with the given service and send it back via `CFP` ACL message.

Listing 4.2: Search in DF.

```java
try {
 DFAgentDescription template = new DFAgentDescription();
 ServiceDescription sd = new ServiceDescription();
 sd.setType("Skill");
 sd.setName(skillType);
 template.addServices(sd);
 DFAgentDescription[] result = DFService.search(this,
  template);
 for (DFAgentDescription agentDesc : result) {
  AID provider = agentDesc.getName();
  ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
  cfp.addReceiver(provider);
  cfp.setContent(productType);
  send(cfp);
 }
} catch (FIPAException fe) {
 LOGGER.log(Level.SEVERE, fe.get);
}
```

A simple way to understand the Product Agent behavior is to imagine that Product A must execute sequence #2, which corresponds to {punch}. Product A agent will interact with the DF agent to implement the yellow pages, where each agent's skills and skill search will be stored. Once found, the agents Punching A and Punching B. The Product Agent sends a message to these two agents, if they are not busy, they return a price, the Product Agent through logic shown in Listing 4.3, checks the lowest price and sends a response message to start the process.

Figure A.4 and A.5 shows a UML diagram with interactions between agents in a JADE environment and displays messages that are exchanged between agents, making it easier to debug and analyze communication patterns within the MAS providing details such as sender, receiver, and message performance.

Listing 4.3: Negotiation between Product Agent and Resource Agent.

```
try {
 int price = Integer.parseInt(parts[1]);
 if (bestSeller == null || price < bestPrice) {
  bestPrice = price;
  bestSeller = msg.getSender();
 }
 offersReceived++;
 addBehaviour(new WaitForOffersBehaviour(2000));
} catch (NumberFormatException e) {
 LOGGER.log(Level.SEVERE, e.getMessage());
}
```

4.2.3 Resource Agent

The Resource Agent's behavior involves cyclically waiting for requests to operate, which the Product Agent assigns after a negotiation procedure.

In this case study, six Resource Agents will be used, corresponding to the two punching machines and two indexed machines from Fischertechnik^{TM}, the IRB 1400 ABB robot manipulator, and the inspection. Table 4.2 summarizes the available skills in each resource.

Table 4.2: Representation of the Resource Agents' Skills.

Resource Agents'	**{Skill}**
Indexed line A	{drill}
Indexed line B	{drill}
Inspection	{inspect}
Manipulator robot	{transfer}
Punching machine A	{punch}
Punching machine B	{punch}

When the part arrives, the Resource Agent changes its state to unavailable and the associated resource executes the proper operation. When the processing is finished, the Resource Agent notifies the Product Agent and after removing the processed part, it becomes again available to execute a new operation.

The sequences of resource skills will be stored in a JSON file whose purpose will be to configure the agents, place their skills in the DF service for each agent, and the MODBUS TCP/IP parameters for communication between JADE and Modicon M340 PLC using the Java Modbus Library (plc4x), which acts as the middleware that enables communication between the JADE agent and physical devices.

The inspection behavior is to check whether the product is in good condition or defective. For this to work, a node-red application was developed with just two buttons, “Good” and “Bad”, which serves to publish a topic on the MQTT Broker and the Inspection Agent subscribe that topic and inform the Product Agent if the product is in perfect condition or defective. In this way, the inspection can assess whether the product meets the quality standards and inform the Product Agent about the inspection status. If the part is defective, corrective action can be taken.

When the IRB 1400 ABB robot manipulator places the product in one of the containers, the product agent will be notified that the part is finished, and it is then up to the product agent to inform the client agent that the product is finished.

The Resource Agent that controls the IRB 1400 ABB robot subscribes to the OPC-UA service that the robot provides. This allows the agent to send commands to the robot and receive status updates. For example, when the Resource Agent detects that a part is ready for transfer, it communicates with the robot over OPC-UA, commanding it to execute the necessary movements.

Figure A.3 shows a UML diagram with interactions between Resource Agent and various components in a MAS. The Resource Agent initiates synchronous communication with the Modicon M340 PLC, which interfaces with physical devices in an industrial environment, using the MODBUS TCP/IP protocol. This is likely for reading or writing data to/from the PLC. The Resource Agent then interacts with the IRB 1400 ABB robot system using the OPC-UA protocol. The Resource Agent subscribes to a topic on the MQTT broker. This allows the Resource Agent to receive messages or updates that are published on this topic. Finally, Node-Red publishes a message to the MQTT broker.

This integration between Resource Agents and physical devices allows the MAS to control the production process efficiently, facilitating complex decision-making and autonomous system behavior based on the current state of the production environment.

The big issue is “Why the use of MODBUS TCP/IP, MQTT, or OPC UA communication protocols ?”. According to [48] most PLCs do not allow agent code, which is typically developed in JAVA, C#, C++, or Python, to run natively. In such a scenario, the agents need to ideally communicate with the PLC at constant intervals, normally such intervals will be longer than the PLC cycle time, and the actual interaction over the network may include several PLC cycles.

The IEEE 2660.1-2020 recommends in this case the use of OPC-UA because it obtained the highest result of the recommended practice for factory automation, therefore it was decided to apply this communication protocol on the IRB 1400 ABB robot. Since MODBUS and MQTT had a relatively better score, it was decided to apply them to the Modicon M340 PLC.

Chapter 5

Experimental Results and Discussion

This chapter provides the experimental results and their discussion of the Small-Scale Production case study using the JADE in the LCAR Manufacturing Cell.

5.1 Practical Results

The case study is ready to be tested and simulated at this stage. This case study will be about agents using JADE, but before actually testing the agents it is necessary to do some practical tests.

The first tests to be carried out were to check whether all Fischertechnik^{TM} devices were working properly. With the Modicon M340 PLC in operation, it was found that many of the sensors and motors were damaged, making it necessary to check where the problems were. Each device is connected to a Printed Circuit Board (PCB) that aims to convert 24V into 5V through an L6225N, in the case of Punching Machine A and Indexed Machine A, the problem was that the L6225N was damaged, and a new one had to be installed.

The IRB 1400 ABB Robot transfers products between machines using suitable RAPID programs, so the next test is to check if the coordinates are correct. The

`PROC()` command was used to define 14 functions (procedures).

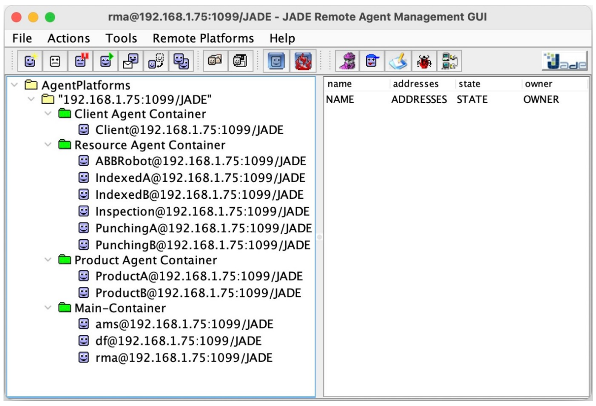
Once the Modicon M340 PLC, the IRB 1400 ABB, and the Fischertechnik^{TM} devices are operational and in perfect condition, conducting tests with the agents is possible. For agents to control Fischertechnik^{TM} devices, Boolean I/Os must be applied to the Modicon M340 PLC in the Ladder system before the first sensors of each device, so that, for example, if the part is placed in the Punching A agent, the sensor `PA_S_T_1` will change to true. Still, only the process will be started with the order of the Punching A agent, that is, the I/0 changes to true until the device does not start the process.

After introducing the agent's boolean I/0s into the ladder system, it is necessary to confirm whether the Modicon M340 PLC is connected to the network via Ethernet. To do this, use the `ping` utility, which uses the ICMP protocol to test connectivity between equipment. Then the MODBUS Coild Register is used to write 1 bit and see if the I/O changes state.

About the IRB 1400 ABB agents, firstly the robot is electrically connected to the Modicon M340 PLC, that is, four PLC outputs are connected to the I/Os of the S4 controller of the IRB 1400 ABB so that at the end of each process they activate an output and the robot moves the part to other positions. Therefore, for MAS to be carried out successfully, there needed to be more outputs on the PLC, but unfortunately, this was not possible, as there were no outputs available. To overcome this obstacle, it was thought that instead of being electrically connected, why not automate the system through Ethernet and `sockets`.

Several communication protocols exist, such as PROFINET, PROFIBUS, MQTT, MODBUS TCP/IP, `sockets`, and OPC-UA. The first to be tested was MODBUS TCP/IP, but through this communication protocol, it is only possible to see the robot's status, that is, whether the robot is on or off and whether it is moving. So MQTT was tested and it didn't work. For a long time, OPC-UA experimented with the help of the article [49, 50, 48] but this is only possible with the IRC5 controller. There is the possibility of implementing the OPC-UA communication protocol in the S4 controller, but for this to work additional installations are required in the robot and Java libraries for this purpose.

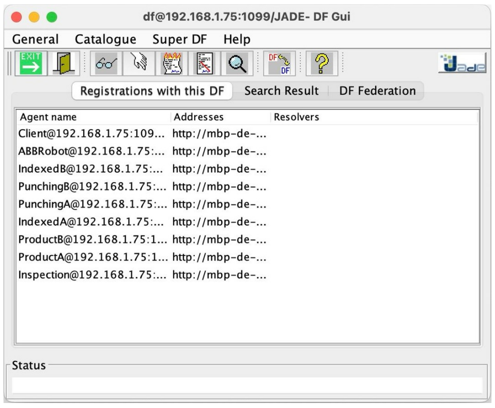
Before starting negotiation tests between agents, it is necessary to check that the agents are initialized and registered in DF. The agents were divided into their containers as illustrated in Figure 5.1a and are registered in the yellow pages as illustrated in Figure 5.1b. Therefore, everything is ready to extract results, which will be covered in the following sections.


rma@192.168.1.75:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help

```
AgentPlatforms
  "192.168.1.75:1099/JADE"
    Client Agent Container
      Client@192.168.1.75:1099/JADE
    Resource Agent Container
      ABBRobot@192.168.1.75:1099/JADE
      IndexedA@192.168.1.75:1099/JADE
      IndexedB@192.168.1.75:1099/JADE
      Inspection@192.168.1.75:1099/JADE
      PunchingA@192.168.1.75:1099/JADE
      PunchingB@192.168.1.75:1099/JADE
    Product Agent Container
      ProductA@192.168.1.75:1099/JADE
      ProductB@192.168.1.75:1099/JADE
    Main-Container
      ams@192.168.1.75:1099/JADE
      df@192.168.1.75:1099/JADE
      rma@192.168.1.75:1099/JADE
```

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

(a) Screenshot of the JADE RMA GUI.



Agent name	Addresses	Resolvers
Client@192.168.1.75:109...	http://mbp-de-...	
ABBRobot@192.168.1.75:...	http://mbp-de-...	
IndexedB@192.168.1.75:...	http://mbp-de-...	
PunchingB@192.168.1.75:...	http://mbp-de-...	
PunchingA@192.168.1.75:...	http://mbp-de-...	
IndexedA@192.168.1.75:...	http://mbp-de-...	
ProductB@192.168.1.75:1...	http://mbp-de-...	
ProductA@192.168.1.75:1...	http://mbp-de-...	
Inspection@192.168.1.75:...	http://mbp-de-...	

(b) Screenshot of the DF GUI.

Figure 5.1: Agents Initialization.

5.2 Negotiation Results

This section aims to illustrate the results obtained from the negotiation between agents to produce Product A and Product B. The term "negotiation" refers to timing the entire process of messages between agents. In this case, Product A refers to the exchange of messages shown in Figure A.4, and Product B refers to the exchange of messages shown in Figure A.5.

In Java, there are several testing frameworks, for example, JUnit, Spock, or TestNG. In the IDE, it's possible to create a test class directly from the source code along with the necessary test methods. It's possible to switch between test classes and source code with a shortcut, run multiple tests, view statistics for each test, and export test results to a file. Therefore, to test the negotiation time, logic was implemented using `System.currentTimeMillis()` that returns the time in milliseconds (ms) of a task as shown in the Algorithm 5.1.

To be able to draw better conclusions from the results, it was chosen to carry out four tests and take the average of the values obtained. Table 5.1 presents the

Algorithm 5.1 Measure Execution Time

```
1: Initialization
2: $startTime \leftarrow$ System.currentTimeMillis()
3: Execute the desired code snippet
4: $endTime \leftarrow$ System.currentTimeMillis()
5: $executionTime \leftarrow endTime - startTime$
6: System.out.println(executionTime)
7: End
```

results obtained for negotiation between agents for Product A, where T corresponds to the number of tests in milliseconds (ms), and Σ corresponds to the time average.

Table 5.1: Negotiation Results for Product A.

Sequence	{Skill}	T_1	T_2	T_3	T_4	Σ
#1	{transfer_1}	2013	2002	2002	2001	2005
#2	{punch}	2007	2001	2001	2007	2004
#3	{transfer_2}	2007	2002	2001	2012	2006
#4	{drill}	2002	2001	2001	2004	2002
#5	{transfer_3}	2013	2002	2002	2001	2005
#6	{inspect}	2004	2001	2009	2014	2007
#7	{transfer_4}	2014	2009	2006	2015	2011

The results of negotiations between agents are all identical and take $\approx$ 2 ms but it is still necessary to discuss them. To discuss the results obtained, it is necessary to transfer them to a graph as shown in Figure 5.2.

The transfer_4 skill presents the highest scores in the T_4 test, followed by T_1, T_2, and T_3. This indicates that the IRB 1400 ABB robot performed less well in the previous skills during the T_4 test. The variability in results suggests that there were inconsistent problems or changes in operating conditions that need to be investigated and corrected.

The skill inspects also performs poorly in T_4 and T_2, with only slight improvement in T_1 and T_3.

In the transfer_2 skill, the test with the worst results is T_4, following the pattern

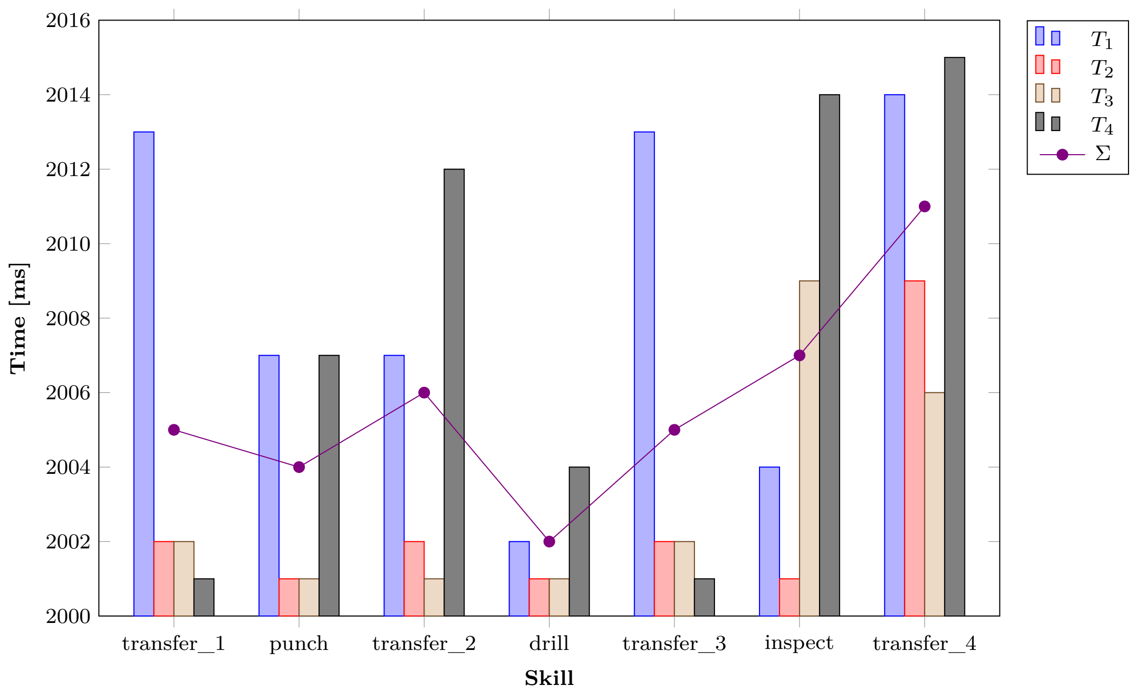


Figure 5.2: Negotiation Results for Product A per Skill.

of the previous skills mentioned. The transfer_1 skill has a similar value for T_2 and T_3, the test with the worst performance was the first one, T_1.

The transfer_3 skill presents the lowest performance observed in T_4, indicating a significant problem in this period; however, there is a noticeable improvement beyond T_2.

The punch skill maintains similar performance in tests 1 and 4, with similar improvement seen in tests 2 and 3. During all periods, the drill skill presents relatively low values and better performance.

Finally, the diagram indicates that the machine underperformed in specific skills during T_4, especially in transfer_3 and inspect. These skills show the highest values and list critical areas that need investigation and improvement. Consistency in abilities like punch suggests stability, but performance is still suboptimal.

For Product B, the timing logic remains the same. Table 5.2 shows the results obtained for Product B. As can be seen, the values did not differ much. Transposing the results into a graph, the result is shown in Figure 5.3.

When negotiating Product B, in the test T_4 still presents similar behavior to Product A, since in seven skills it performs worse in five.

Table 5.2: Negotiation Results for Product B.

Sequence	**{Skill}**	T_1	T_2	T_3	T_4	Σ
#1	{transfer_1}	2001	2001	2001	2015	2005
#2	{drill}	2001	2001	2001	2008	2003
#3	{transfer_2}	2001	2003	2000	2013	2004
#4	{punch}	2001	2001	2001	2007	2003
#5	{transfer_3}	2001	2001	2001	2015	2005
#6	{inspect}	2001	2015	2013	2003	2008
#7	{transfer_4}	2001	2003	2008	2006	2005

In the inspect skill, the T_1 test has great performance and T_2 is the test with the worst average values, because sometimes the Inspection Node-Red did not submit the topic immediately, perhaps because of the network.

In transfer_1 and transfer_3 there are excellent values in all tests besides T_4. A pattern that was not observed until skill transfer_4 is that T_3 is the test with the highest average values in a skill. The opposite happens in transfer_2, that is, T_3 presents the lowest average value.

In the skill drill and punch, the tests T_1, T_2, and T_3 present equal results, showing consistency. The test with the worst performance, but also with similar results in both skills was T_4. Finally, it was interesting to observe that T_1 presents the same average results in all skills.

Neither product significantly stands out over the other in terms of overall performance, as they both have similar strengths.

However, given that punch is a frequently used skill in many production processes and considering overall performance, Product B would be the best choice for developing a MAS to control LCAR cell. The excellent punch performance suggests that Product B would be more efficient in critical and repetitive operations common in automated manufacturing environments.

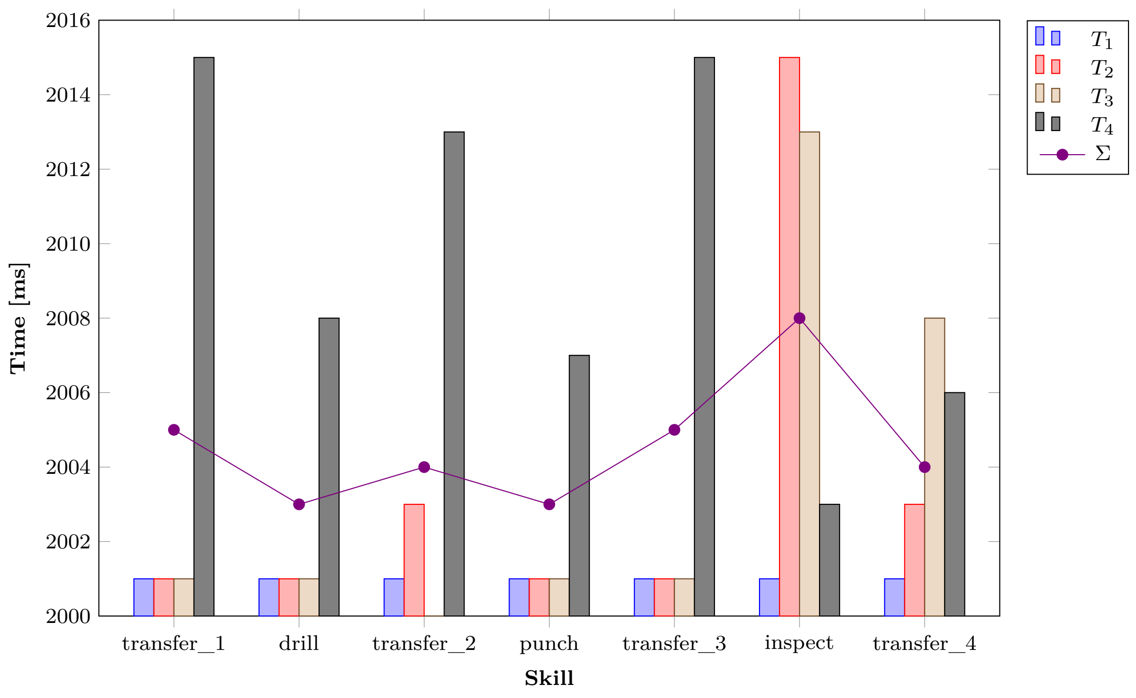


Figure 5.3: Negotiation Results for Product B per Skill.

5.3 Process Plans Results

This section aims to illustrate the results obtained and discuss them to visualize how the system behaves to produce a certain product.

To produce a product there are certain processes to be carried out as previously illustrated in Table 4.1.

Table 5.3 shows the data obtained for the production of Product A up to five quantities, where T is the number of tests in milliseconds (ms), Q is the number of quantities, and Σ corresponds to the average for each quantity.

Table 5.3: Process Plans Results for Product A.

Test	Q_1	Q_2	Q_3	Q_4	Q_5
T_1	76611	140030	201555	261869	323972
T_2	70889	142364	208638	270955	334483
T_3	71728	135338	196130	258246	319466
T_4	84957	146810	211990	275623	337580
Σ	76046	141136	204578	266673	328875

It can be seen that to produce one quantity of Product A it takes ≈ 76046 ms, that is ≈ 1.26 min and to produce five quantities it takes ≈ 326252 ms, that is, ≈ 5.48 min. This is a series with an additive trend and additive seasonality, as shown in Figure 5.4.

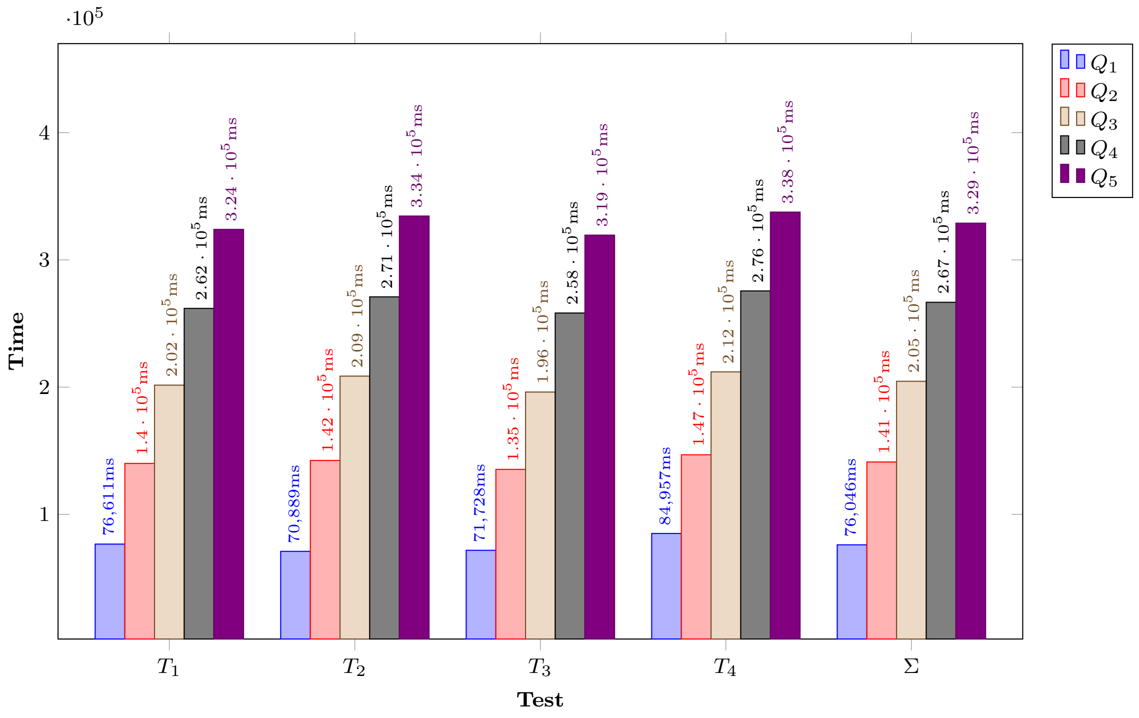


Figure 5.4: Process Results for Product A per Quantity.

As mentioned previously the quantity values increase consistently with increasing quantity at each stage.

Indicating that the quantity is 5, it has a higher average value, and the respective columns (Q_5) are the largest in all stages.

The commitments do not seem to vary much in the same amount, demonstrating a consistency in values between advances.

The values of each quantity do not change significantly between steps. For example, Q_5 is high and consistent across all phases. This may suggest that the variable being monitored or the process itself acts consistently throughout the various stages.

It can be deduced, for example, that in a production process, the increase in the quantity produced causes the average value generated to increase predictably.

To produce Product B, the same thing happens as Product A, depending on the quantity, the time increases linearly as shown in Table 5.4 to produce a quantity of

Product B, it takes ≈ 81919 ms, that is ≈ 1.36 min and to produce five quantities ≈ 357904 ms are needed, that is, ≈ 5.96 min.

Table 5.4: Process Plans Results for Product B.

Test	$\boldsymbol{Q_1}$	$\boldsymbol{Q_2}$	$\boldsymbol{Q_3}$	$\boldsymbol{Q_4}$	$\boldsymbol{Q_5}$
T_1	76476	141066	222419	298180	350177
T_2	91455	156443	221459	310544	377304
T_3	73732	137773	207899	267787	329163
T_4	86012	178591	243987	309690	374972
Σ	81919	153468	223941	296550	357904

The values increase consistently in each test (T_1, T_2, T_3, and T_4) as quantities increase from 1 to 5, as shown in Figure 5.5. This suggests that larger values at all stages result from larger quantities. The values of Q_5 in T_1 are noticeably higher

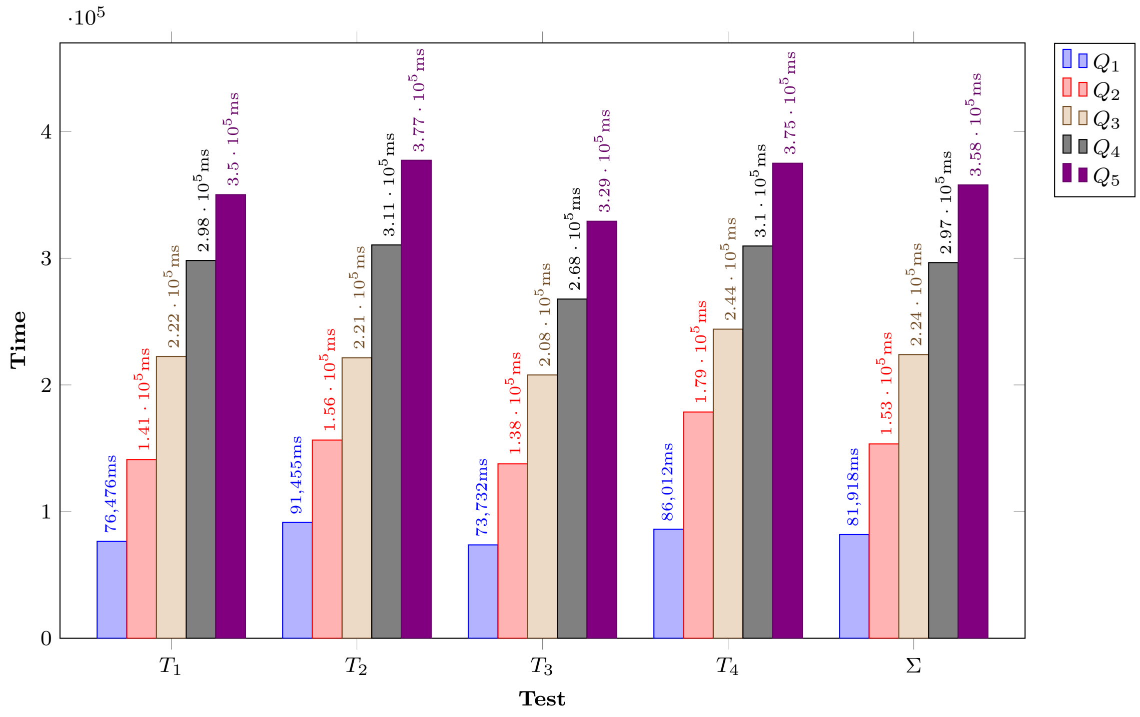


Figure 5.5: Process Results for Product B per Quantity.

than in the other stages, which may point to a specific variation or occurrence that increased the average values in this stage.

The values of each quantity become more aligned in subsequent tests (T_2, T_3, and T_4). Test 1 presents a significantly higher value for Q_5, which may indicate an efficiency peak or an anomaly that resulted in a higher value.

To conclude this section, focusing on the accuracy and representatives of the data, the first graph (Figure 5.4) may be more suitable if wanted to show a more uniform and consistent progression in the test results. The second graph (Figure 5.5) may be more suitable if interested in values that start lower and have a greater variation between quantities and tests.

Considering the performance analysis of the IRB 1400 ABB Robot, the first graph (Figure 5.4) seems to provide a more balanced and consistent view of the test results. The second graph (Figure 5.5) shows greater variations, which can be useful for highlighting specific discrepancies between different tests and quantities.

Chapter 6

Conclusions and Future Work

This chapter will conclude this dissertation by reviewing the work presented. Finally, it will suggest possible areas of future work that could improve and expand the work carried out here.

Industry 4.0 introduces several transformations in industry and other sectors, particularly driven by digital transformation.

It can be said that MAS has evolved and shown clear signs of clear signs that they have enormous computational potential in Industry 4.0 and CPS.

The application of agents has been growing rapidly, which leaves room for the emergence of varied MAE, requiring standardization to maintain a degree of interoperability.

The need for frameworks that help development, with JADE as an excellent reference. This is because JADE offers a robust, secure, concise environment and abstracts the agent developer from the need to worry about the need to worry about implementing an efficient agent platform, communication, message exchange, and many other attributes that a multi-agent system needs.

As CPS are systems characterized by the combination of cyber and physical counterparts, constituting the backbone for implementing the principles of Industry 4.0, this dissertation served to enter the worlds of DAI, CPS and agents. The case study presented here served to familiarize with agents, the types of agents, MAS, apply engineering concepts in LCAR such as communication protocols, and get started in research, as it took a lot of research to achieve the objectives.

At first, it wasn't easy to understand the meaning of the agent and the Java programming language, but after some research, it was possible to overcome this problem.

However, in this case study it is necessary to make some modifications to be 100% effective that will be broken in the future, such as the possibility of requesting multiple products, for example, while the system is producing Product A, request a Product B and see how MAS behaves; continue with the OPC-UA investigation to apply to the IRB 1400 ABB robot; if it is possible to add Fischertechnik^{TM} device to introduce more process sequences and thus create more agents of the Product type; and finally implement the same MAS in other frameworks to check which is the best framework for running agents.

Bibliography

[1] D. G. Fabio Luigi Bellifemine, Giovanni Caire, *Developing Multi-Agent Systems with JADE.* John Wiley & Sons, Ltd, 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470058411.fmatter

[2] MODBUS, "Modbus application protocol specification v1.1b3 april," , 2012. [Online]. Available: http://www.modbus.org1/50

[3] Service sets. [Online]. Available: https://reference.opcfoundation.org/Core/Part4/v104/docs/5

[4] Mappings. [Online]. Available: https://reference.opcfoundation.org/Core/Part6/v105/docs/

[5] Christine, "How does industry 4.0 differ from the previous generation?" 2024. [Online]. Available: https://www.renaix.com/industry-4-0-the-fourth-industrial-revolution/

[6] B. Bordel Sánchez, R. Alcarria, T. Robles, and D. Martín, "Cyber-physical systems: Extending pervasive sensing from control theory to the internet of things," *Pervasive and Mobile Computing*, vol. 40, 06 2017.

[7] A. JANOTA, "Collective artificial intelligence multiagent systems artificial intelligence," 2023.

[8] H. Hatteland and O. Fleckenstein, "Multi-agent systems," *Bachelor thesis, Technical University of Denmark*, 2017.

[9] Y. Rizk, M. Awad, and E. W. Tunstel, “Decision making in multiagent systems: A survey,” *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 3, pp. 514–529, 2018.

[10] P. G. Balaji and D. Srinivasan, “An introduction to multi-agent systems,” *Innovations in multi-agent systems and applications*, 2010.

[11] G. Kanagachidambaresan, R. Anand, E. Balasubramanian, and V. Mahima, *Internet of Things for industry 4.0.* Springer, 2020.

[12] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture.* Springer Science & Business Media, 2009.

[13] O. Palonen *et al.*, “Opc ua-tietomallien oliopohjainen toteutus java-kielellä,” Master’s thesis, aalto university, 2010.

[14] P. Leitão, A. W. Colombo, and S. Karnouskos, “Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges,” *Computers in Industry*, vol. 81, pp. 11–25, 2016, emerging ICT concepts for smart, safe and sustainable industrial systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361515300348

[15] E. A. Lee, “Cyber physical systems: Design challenges,” in *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC).* IEEE, 2008, pp. 363–369.

[16] L. C. Jain and R. K. Jain, *Hybrid intelligent engineering systems.* World Scientific, 1997, vol. 11.

[17] H.-N. L. Teodorescu, A. Kandel, and L. C. Jain, *Fuzzy and neuro-fuzzy systems in medicine.* CRC Press, 1998, vol. 2.

[18] N. Ichalkaranje, L. C. Jain, R. Khosla, and S. Pierre, “Using stationary and mobile agents for information retrieval and e-commerce,” *Design of Intelligent Multi-Agent Systems: Human-Centredness, Architectures, Learning and Adaptation*, pp. 387–448, 2005.

[19] G. Resconi and L. Jain, *Intelligent agents.* Springer, 2004.

[20] G. Weiss *et al.*, "Multiagent systems (intelligent robotics and autonomous agents series)," 2013.

[21] J. Tweedale, N. Ichalkaranje, C. Sioutis, B. Jarvis, A. Consoli, and G. Phillips-Wren, "Innovations in multi-agent systems," *Journal of Network and Computer Applications*, vol. 30, pp. 1089–1115, 8 2007.

[22] N. Vlassis, *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence.* Springer Nature, 2022.

[23] D. S. Weld, C. H. Lin, and J. Bragg, *Artificial intelligence and Collective Intelligence.* MIT Press, 2015.

[24] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, pp. 345–383, 2000.

[25] Z. Ren and C. J. Anumba, "Learning in multi-agent systems: a case study of construction claims negotiation," *Advanced engineering informatics*, vol. 16, no. 4, pp. 265–275, 2002.

[26] C. V. Goldman, "Learning in multi-agent systems," in *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, 1996, pp. 1363–1363.

[27] F. Bergenti and A. Ricci, "Three approaches to the coordination of multiagent systems," in *Proceedings of the 2002 ACM symposium on Applied computing*, 2002, pp. 367–372.

[28] T. C. Hsia and M. Soderstrand, "Development of a micro robot system for playing soccer games," in *Proceedings of the Micro-Robot World Cup Soccer Tournament*, 1996, pp. 149–152.

[29] P. Balaji and D. Srinivasan, "Distributed multi-agent type-2 fuzzy architecture for urban traffic signal control," in *2009 IEEE International Conference on Fuzzy Systems.* IEEE, 2009, pp. 1627–1632.

[30] L. E. Parker, “Heterogeneous multi-robot cooperation,” , 1994.

[31] L. Skorin-Kapov, M. Mosmondor, O. Dobrijevic, and M. Matijasevic, “Application-level qos negotiation and signaling for advanced multimedia services in the ims,” *IEEE Communications Magazine*, vol. 45, no. 7, pp. 108–116, 7 2007.

[32] K. Trzec and I. Lovrek, “Field-based coordination of mobile intelligent agents: an evolutionary game theoretic analysis,” in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems.* Springer, 2007, pp. 198–205.

[33] A. K. Agogino and K. Tumer, “Team formation in partially observable multi-agent systems,” in *International Joint Conference on Neural Networks*, 2004.

[34] Y. Shoham, R. Powers, and T. Grenager, “If multi-agent learning is the answer, what is the question?” *Artificial Intelligence*, vol. 171, no. 7, pp. 365–377, 2007, foundations of Multi-Agent Learning. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370207000495

[35] K.-Y. Kim, L. Monplaisir, and J. Rickli, *Flexible Automation and Intelligent Manufacturing: The Human-Data-Technology Nexus: Proceedings of FAIM 2022, June 19–23, 2022, Detroit, Michigan, USA, Volume 2.* Springer Nature, 2023.

[36] P. F. Oliveira, P. Novais, and P. Matos, “Consumption performance, of a multi agent system used to achieve comfort preferences,” in *International Symposium on Ambient Intelligence.* Springer, 2023, pp. 199–208.

[37] P. Leitão, J. Barbosa, G. S. Funchal, and V. Melo, “Self-organized cyber-physical conveyor system using multi-agent systems,” *International Journal of Artificial Intelligence*, 2020.

[38] J. Barbosa and P. Leitão, “Simulation of multi-agent manufacturing systems using agent-based modelling platforms,” in *2011 9th IEEE International Conference on Industrial Informatics.* IEEE, 2011, pp. 477–482.

[39] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa *et al.*, "Jade-a white paper. exp in search of innovation," in *EXP in search of innovation.* Telecom Italia, 2003, vol. 3, pp. 6–19.

[40] Ö. Özkan, "Modelling a university as a multi-agent system," Master's thesis, Fen Bilimleri Enstitüsü, 2008.

[41] FIPA, "Fipa acl message structure specification." [Online]. Available: http://www.fipa.org/specs/fipa00061/SC00061G.html#_Toc26669700

[42] ——, "Interaction protocol specifications." [Online]. Available: http://www.fipa.org/repository/ips.php3

[43] S. Poslad, "Specifying protocols for multi-agent systems interaction," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 2, 11 2007.

[44] "Mqtt." [Online]. Available: https://mqtt.org/faq/

[45] F. Pauker, S. Wolny, S. M. Fallah, and M. Wimmer, "Uml2opc-uatransforming uml class diagrams to opc ua information models," *Procedia CIRP*, vol. 67, pp. 128–133, 2018.

[46] Specification parts. [Online]. Available: https://reference.opcfoundation.org/Core/Part1/v105/docs/4.2

[47] J. Banks, "Introduction to simulation," in *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, 1999, pp. 7–13.

[48] P. Leitão, T. I. Strasser, S. Karnouskos, L. Ribeiro, J. Barbosa, and V. Huang, "Recommendation of best practices for industrial agent systems based on the ieee 2660.1 standard," in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, vol. 1, 2021, pp. 1157–1162.

[49] J. Dias, J. Vallhagen, J. Barbosa, and P. Leitão, "Agent-based reconfiguration in a micro-flow production cell," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, 2017, pp. 1123–1128.

[50] P. Reguera, M. Domínguez, H. Alaiz, J. Fuertes, M. Prada, and R. García, "Remote operation of abb irb 1400 s4 robot over the internet," *IFAC Proceedings Volumes*, vol. 39, no. 6, pp. 505–510, 2006.

Appendix A

UML Diagrams

The standardized modeling language known as UML offers a thorough foundation for describing, creating, visualizing, and documenting software system artifacts. This appendix explores the different kinds of UML diagrams.

UML diagrams are vital tools in software engineering that help communicate and comprehend the functioning and design of the system. The appendix demonstrates how UML may describe interactions between various system components and external actors by providing extensive sequence diagrams for agents.

The appendix begins by exploring the FIPA-ACL interactions between the Client Agent and Product Agent as shown in Figure A.1, and between the Product Agent and Resource Agent as shown in Figure A.2 focusing on the request, confirmation, and refusal processes that define their communication. This is followed by a detailed analysis of the Product Agent interactions between DF highlighting the processes involved in service discovery, negotiation, and task allocation.

It then explores the interactions between Resource-type agents with devices such as the Modicon M340 PLC using the MODBUS TCP/IP protocol with Read/Write Access and the IRB 1400 ABB Robot using the OPC-UA protocol as shown in Figure A.3.

The interaction sequences for Product A and Product B are shown visually in the figures in this appendix, such as Figure A.4 and Figure A.5, respectively. These diagrams exemplify the methodical approach that UML provides for modeling software activity by showing the flow of requests and answers among different agents.

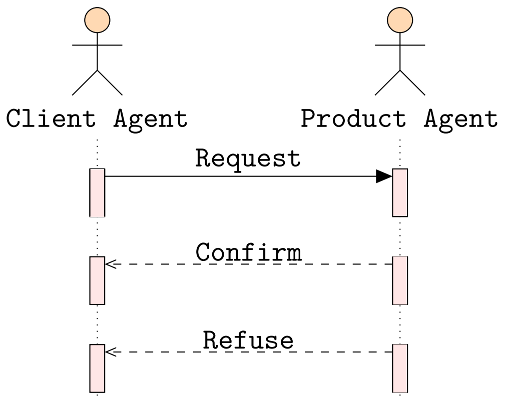


Figure A.1: UML Diagram between Client Agent and Product Agent.

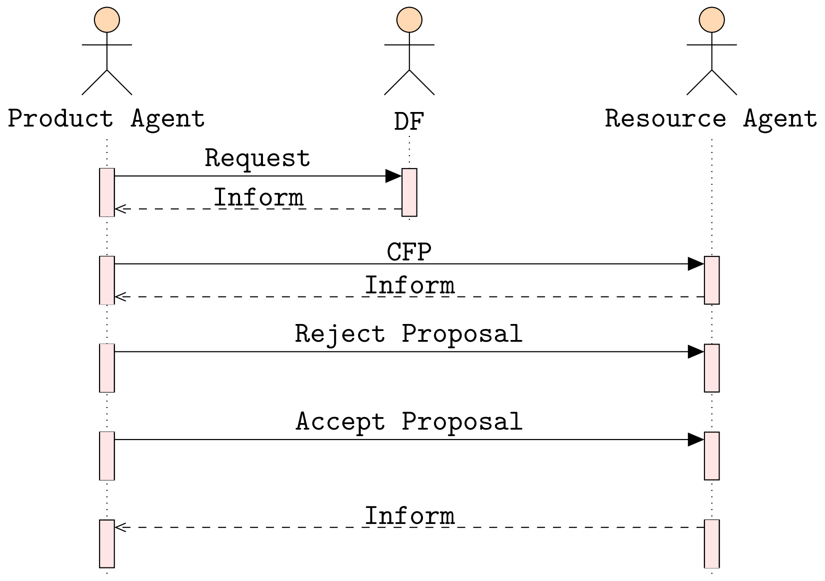


Figure A.2: UML Diagram between Product Agent and Resource Agent.

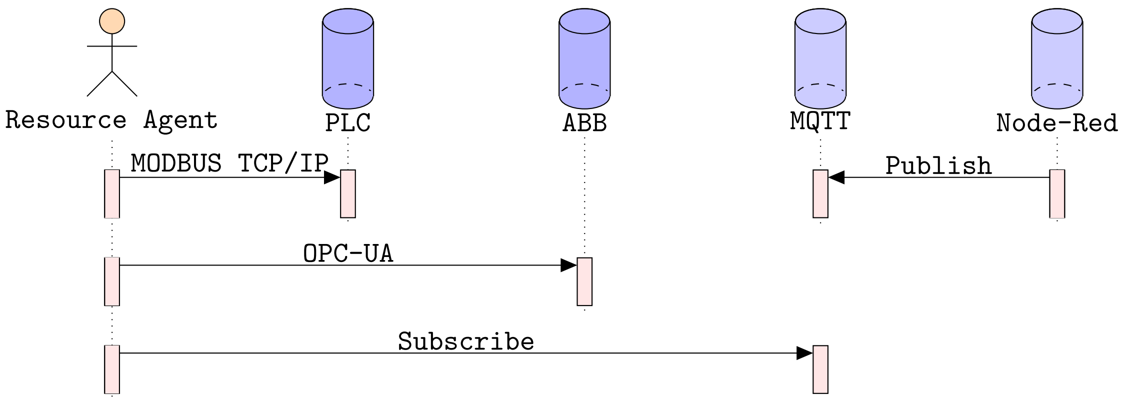


Figure A.3: Resource Agent UML Diagram.

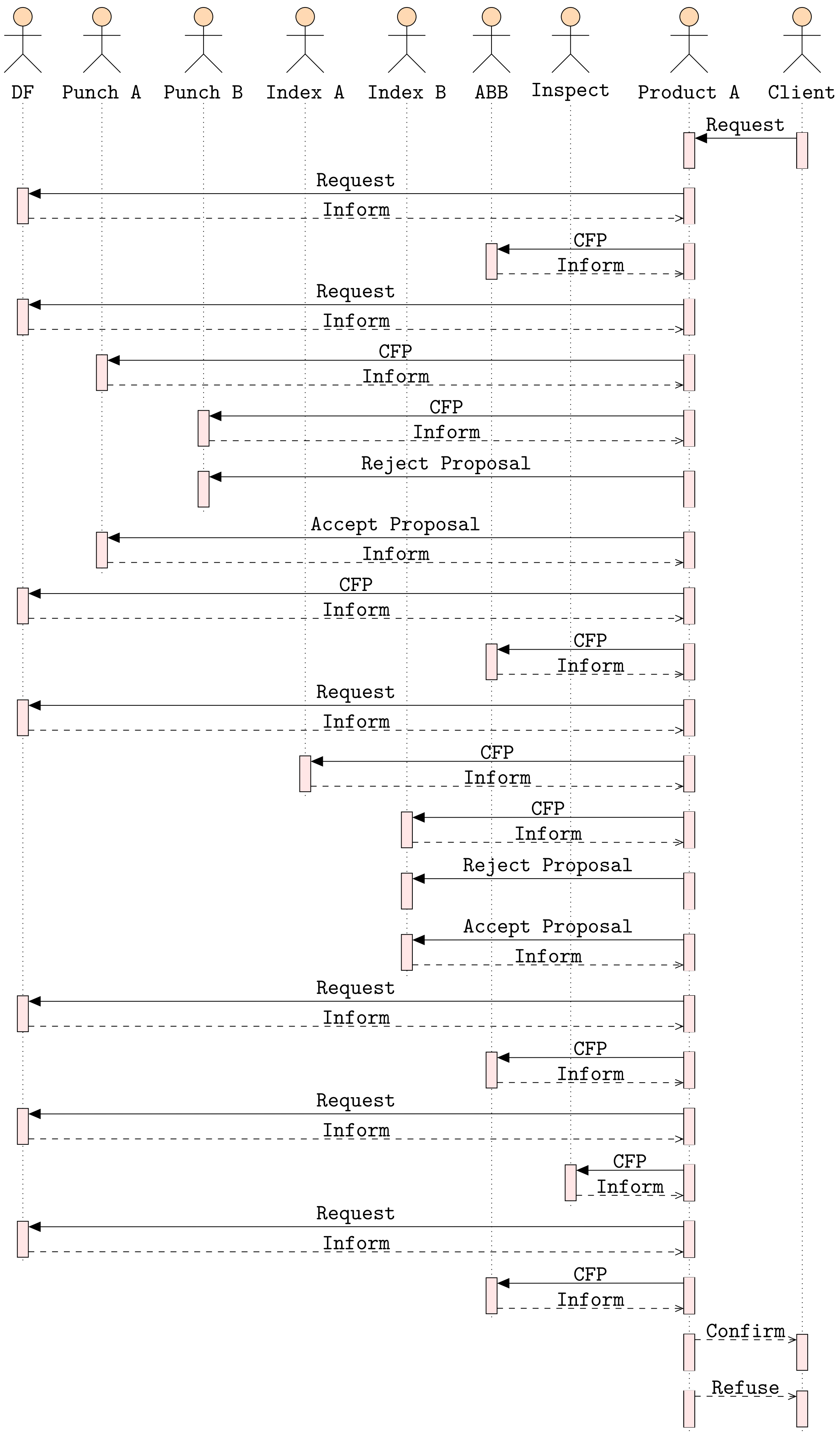


Figure A.4: Product A UML Diagram.

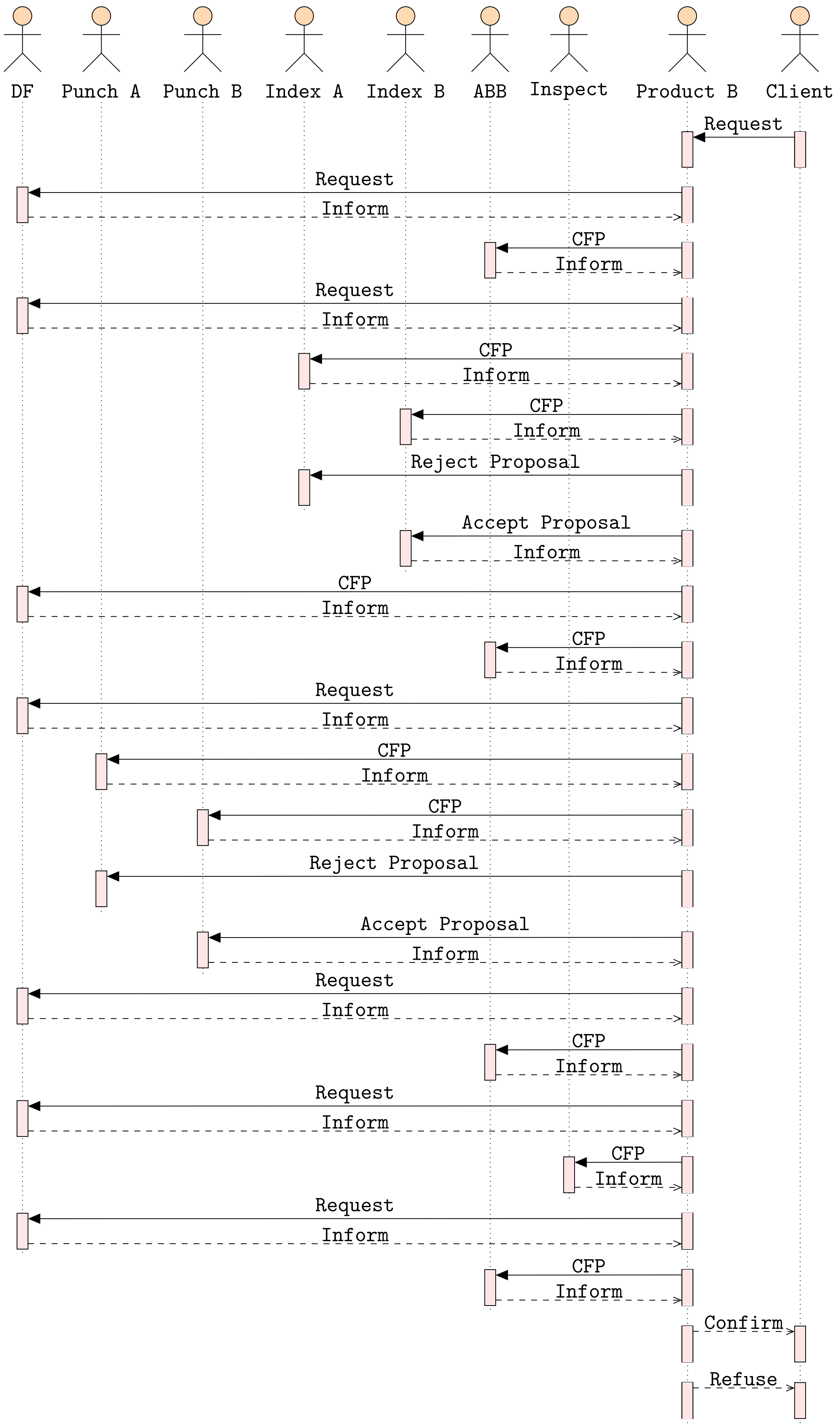


Figure A.5: Product B UML Diagram.

Appendix B

Source Code Developed in this Work

This appendix provides the code files used in this dissertation for used to perform a MAS for the LCAR. The full code is available at GitHub link.

First, before talking about the agent code, to avoid always creating new Java Command-Line Interface (CLI), a file called `Main.java` was created which contains the commands to make `createNewAgent`, the file `MainContainer.java` includes the code to run the GUI of the JADE RMA, and finally the file `JadeContainer.java` contains the code to divide the agents by your containers, instead of placing all the agents in the same container or the main container.

So everything is ready to talk about the files for each of the agents, which are:

- Client Agent where the following codes were made:
  - `src/main/java/factory/Clientgui.form` which contains the graphic design for the Client to request products;
  - `src/main/java/factory/Clientgui.java` contains the java code for the agent;
- Product Agent where the following codes were made:
  - `ProductA.json`, JSON file to configure the Product A agent of type Product Agent;

- `ProductB.json`, JSON file to configure the Product B agent of type Product Agent;
- `src/main/java/factory/Product.java`, contains the java code for the agent;

- Finally, Resource Agent where the following codes were made:
  - `ABBRobot.json`, JSON file to configure the IRB 1400 ABB robot agent of type Resource Agent;
  - `ABBRobot-RAPID.PRG`, a RAPID program with the coordinates used to control the IRB 1400 ABB robot;
  - `Fischertechnik.auto.auto.auto.sta`, ladder program to control the Fischertechnik^{TM} devices and configuration for MODBUS TCP/IP Protocol;
  - `IndexedA.json`, JSON file to configure the Indexed A agent of type Resource Agent;
  - `IndexedB.json`, JSON file to configure the Indexed B agent of type Resource Agent;
  - `Inspection.json`, JSON file to configure the Inspection agent of type Resource Agent;
  - `PunchingA.json`, JSON file to configure the Punching A agent of type Resource Agent;
  - `PunchingB.json`, JSON file to configure the Punching B agent of type Resource Agent;
  - `src/main/java/factory/Resource.java`, contains the java code for the Resource Agent.

The communication between Resource type agents is through MODBUS TCP/IP, using the library ttfamily plc4x, using the following codes, the files of which are:

- `src/main/java/modbus/ModbusIA`, file to write to I/O boolean M83, which corresponds to the Indexed A agent;

- `src/main/java/modbus/ModbusIB`, file to write to I/O boolean M84, which corresponds to the Indexed B agent;
- `src/main/java/modbus/ModbusPA`, file to write to I/O boolean M81, which corresponds to the Punching A agent;
- `src/main/java/modbus/ModbusPB`, file to write to I/O boolean M82, which corresponds to the Punching B agent;
- Finally, `src/main/java/modbus/ReadPLC`, file to read the I/Os of the Modicon M340 PLC.